

AD-A081 037

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/6 9/2

A PORTABLE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE PACKAGE--ETC(U)

SEP 78 H J ROOD

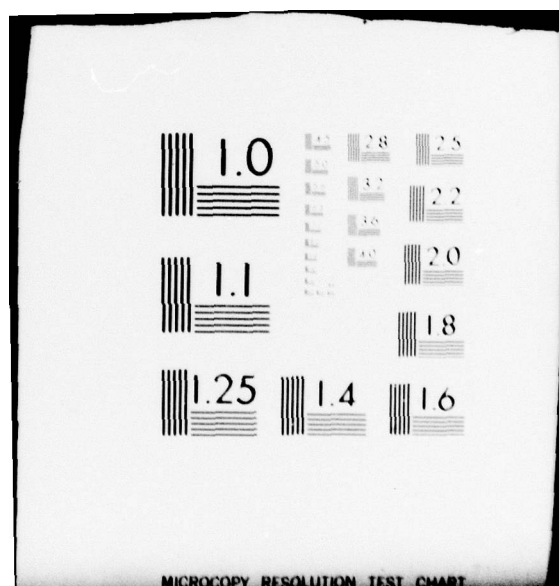
UNCLASSIFIED

NL

1 OF 3

AD  
A081037





DDC FILE COPY ADA081037

LEVEL

(2)  
NU

# NAVAL POSTGRADUATE SCHOOL

Monterey, California



DTIC  
ELECTE  
FEB 25 1980  
S D A

## THESIS

A PORTABLE THREE-DIMENSIONAL GRAPHICS SOFTWARE  
PACKAGE

by

Homer John Rood, Sr.

September 1978

Thesis Advisor:

M. L. Cotton

Approved for public release; distribution unlimited

THIS DOCUMENT IS BEST QUALITY PRACTICABLE.  
THE COPY FURNISHED TO DDC CONTAINED A  
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.

80 2 19 106

## **DISCLAIMER NOTICE**

**THIS DOCUMENT IS BEST QUALITY  
PRACTICABLE. THE COPY FURNISHED  
TO DDC CONTAINED A SIGNIFICANT  
NUMBER OF PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.**

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle)		5. TIME OF REPORT & PERIOD COVERED
A PORTABLE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE PACKAGE		Master's Thesis September 1978
6. PERFORMING ORG. REPORT NUMBER		7. CONTRACT OR GRANT NUMBER(s)
8. AUTHOR		9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
Homer John Rood, Sr.		
10. PERFORMING ORGANIZATION NAME AND ADDRESS		11. REPORT DATE
Naval Postgraduate School Monterey, California 93940		1 Sep 1978
12. CONTROLLING OFFICE NAME AND ADDRESS		13. NUMBER OF PAGES
Naval Postgraduate School Monterey, California 93940		270
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report)
12 272		Unclassified
16. DISTRIBUTION STATEMENT (of this Report)		18a. DECLASSIFICATION/DOWNGRADING SCHEDULE
Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Three-dimensional graphics; Hidden line elimination; Hidden surface elimination.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
A portable three-dimensional computer graphics software package was developed utilizing the Fortran language. The package included the capability of displaying any object as a wire-frame image, as a wire-frame image with hidden lines removed,		

or as a solid figure with hidden surfaces removed. This computer graphics package provides the user with the ability to rotate, scale, and translate any part of the displayable image. It was utilized to display images on four distinct display devices with only minor software alterations. Three totally different host computers supported the four display devices.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	23 CIR

Approved for public release; distribution unlimited

A PORTABLE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE  
PACKAGE

by

Homer J. Rood, Sr.  
Lieutenant, United States Navy  
E.S., United States Naval Academy, 1972

Submitted in partial fulfillment of the  
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the  
NAVAL POSTGRADUATE SCHOOL  
September 1978

Author:

*Homer J. Rood*

Approved by:

*Mitchell A. Cotton*

THESIS ADVISOR

*R. Paulholzer*

SECOND READER

*W. K. Kik*

Chairman, Department of Electrical Engineering


*William M. Toller*

Dean of Science and Engineering



## ABSTRACT

A portable three-dimensional computer graphics software package was developed utilizing the Fortran language. The package included the capability of displaying any object as a wire-frame image, as a wire-frame image with hidden lines removed, or as a solid figure with hidden surfaces removed. This computer graphics package provides the user with the ability to rotate, scale, and translate any part of the displayable image. It was utilized to display images on four distinct display devices with only minor software alterations. Three totally different host computers supported the four display devices.



## TABLE OF CONTENTS

I.	INTRODUCTION.....	17
II.	THREE-DIMENSIONAL GRAPHICS.....	19
	A. DISPLAY OF AN OBJECT AND ITS MOTION.....	19
	B. IMAGE REALISM.....	25
	C. GRAPHICS SOFTWARE STRUCTURE.....	29
III.	GRAPHICS SOFTWARE COMPONENTS.....	33
	A. OBJECT COORDINATE SYSTEM -THE DATA BASE.....	33
	B. IMAGE SCALING.....	34
	C. TRANSLATION.....	34
	D. IMAGE ROTATION.....	35
	1. X-Axis Rotation.....	35
	2. Y-Axis Rotation.....	35
	3. Z-Axis Rotation.....	36
	4. An Arbitrary Axis of Rotation.....	36
	E. COORDINATE TRANSFORMATIONS.....	41
	1. Object to Eye Coordinate Transformation..	41
	2. Eye to Screen Coordinate Transformation..	48
	F. DISPLAY CLIPPING.....	50
	G. HIDDEN LINE REMOVAL.....	54
	H. HIDDEN SURFACE REMOVAL.....	69
	I. IMAGE SHADING.....	84
IV.	HARDWARE AND SOFTWARE CONSIDERATIONS.....	85
	A. IMAGE DISPLAYED - ALL LINES SHOWN.....	85
	B. INTERACTIVE SOFTWARE AND HARDWARE.....	88
	C. HIDDEN LINE REMOVAL.....	91
	D. HIDDEN SURFACE REMOVAL.....	93
V.	A THREE-DIMENSIONAL GRAPHICS APPLICATION.....	95
	A. PLANAR SURFACE PENETRATION.....	96
	E. HIDDEN SURFACES DISPLAYED.....	97
	C. TORPEDO TEST AREA SIMULATION.....	100

VI. CONCLUSIONS.....	105
Appendix A: THREE-DIMENSIONAL GRAPHICS SUERCUTINES AND FLOW CHARTS.....	107
Appendix B: APPLICATIONS PROGRAM AND SUBROUTINES.....	251
FOOTNOTES.....	268
LIST OF REFERENCES.....	269
INITAL DISTRIBUTION LIST.....	270

## LIST OF FIGURES

1.	TWO-D ROTATION OF A POINT.....	23
2.	EYE COORDINATE SYSTEM.....	24
3.	WIRE-FRAME IMAGE.....	27
4.	HIDDEN LINES REMOVED.....	28
5.	N BY M DOT MATRIX SCREEN REPRESENTATION.....	31
6.	DISPLAY SURFACE DIMENSIONS.....	32
7.	ROTATION INTO X-Z PLANE.....	39
8.	ROTATION INTO Z-AXIS.....	40
9.	EYE COORDINATE SYSTEM.....	42
10.	VIEWING TRANSFORMATION.....	44
11.	Z -AXIS ROTATED TOWARDS ORIGIN..... e	47
12.	PROJECTION OF DISPLAY POINTS ONTO SCREEN.....	49
13.	CLIPPING COORDINATES.....	51
14.	DISPLAY SCREEN DIVISION AND CODING.....	52
15.	POLYGONAL CLASSIFICATIONS.....	59
16.	EDGES'S ANGULAR COMPUTATION.....	60
17.	DISPLAY AREA DIVISION AND CODING.....	61
18.	ANGLE MAGNITUDE OF FOUR.....	64
19.	EDGE DIVISION FOR PROPER ANGLE COMPUTATION.....	65
20.	DEPTH COMPUTATION AT FOUR CORNERS OF WINDOW.....	66

21.	POLYGONAL PENETRATION.....	67
22.	HIDDEN LINE REMOVAL FLOW CHART.....	68
23.	SCAN LINE INTERSECTION OF A POLYGON.....	71
24.	SEGMENT CLASSIFICATION.....	72
25.	ENTERING AND DEPARTING SEGMENTS.....	76
26.	UPDATE OF SEGMENT BLOCK STORAGE.....	77
27.	SPAN DESCRIPTIVE TERMS FOR A SEGMENT.....	79
28.	SEGMENT BOX DEFINITION.....	80
29.	IMPLIED EDGE GENERATION.....	82
30.	HIDDEN SURFACE REMOVAL FLOW CHART.....	83
31.	VIEWPOINT REQUIRED TO DISPLAY THE BACK SURFACES....	99
32.	TORPEEC TEST AREA.....	103
33.	TORPEDC APPROXIMATION.....	104

## LIST OF SYMBOLS

### 1. Display a Wire-Frame Image Symbols

A	distance of the viewer from the display area
B	vertical dimension of the display area
EDGE	array containing the indices of both endpoints of each edge
EDGE1	array containing the indices of the first endpoint of each edge
EDGE2	array containing the indices of the second endpoint of each edge
EDGEN	the number of screen coordinate edges
EDGEN	the number of object coordinate edges
ICHK	array used to code location of two endpoints of an edge being clipped
IR	array used to define the part of the image to be transformed
FECHANG	array used to mark points which have been transformed
FOINTN	the number of screen coordinate points
FOINTN	the number of object coordinate points
FOIGN	the number of polygons
POLYGN	array containing indices of the edges which describe each edge
POLYHE	array containing indices of the polygons which describe each polyhedron
SHAD	array containing the shades or colors of each polygon
VCX	Xs value of the screen center
VCY	Ys value of the screen center

VSX	x resolution divided by 2
VSX	y resolution divided by 2
VX	x object coordinate value of the viewpoint
VY	y object coordinate value of the viewpoint
VZ	z object coordinate value of the viewpoint
XE	array of x object coordinate values of the vertices
XS	array of x screen coordinate values of the vertices
YE	array of y object coordinate values of the vertices
YS	array of y screen coordinate values of the vertices
ZE	array of z object coordinate values of the vertices
ZS	array of z screen coordinate values of the vertices

## 2. Hidden Line Removal Symbols

DELTAT	the incremental angle of each edge
EDLINK	array containing the linked list of edges for each polygon
HIDER	the index of the surrounder polygon closest to the viewer
IBOTT	the Ys value of the bottom of the display area
ISIZEY	the number of resolution elements per vertical line of the display surface
ISIZEY	the number of verticle lines of resolution of the display surface
ISTACK	array used to stack the display window dimensicns
ISTPTR	pointer to last display window added to the stack
JT1	the index of the first point of a polygon's

	edge found by GETNEX
JT2	the index of the second point of a polygon's edge found by GETNEX
LEFT	the Xs value of the left edge of the display area
NEXTED	the index of the next edge for a polygon
OLDP	the index of the last polygon
F	the index of the current polygon
FOLEDG	array containing the index of the first edge of each polygon
POLINK	array containing the linked list of intersector and surrounders
PCLPTR	index of the first polygon
PCLYA	array of x value coefficients for each pplygonal plane
POLYB	array of y value coefficients for each pplygonal plane
PCLYC	array of z value coefficients for each pplygonal plane
POLYD	array of coefficients of the polygonal plane's constant
PCLZMN	the point of each polygon closest to the viewer
SURRND	index of the first polygon on the surround list
THETA	the total angle of the current polygon for this display window
WBY	Ys value of the bottom of the display area
WLX	Xs value of the left-hand edge of the display area
WRX	Xs value of the right-hand edge of the display area
WTY	Ys value of the top edge of the display area
YN	Xs value of the first vertex of the current edge
XP	Xs value of the second vertex of the current

	edge
XSS	array of Xs coordinate values of display vectors
YN	Ys value of the first vertex of the current edge
YF	Ys value of the second vertex of the current edge
YSS	array of Ys coordinate values of display vectors
ZNAX1	the maximum left-lower Zs value of surrander polygons for the current display window
ZNAX2	the maximum left-upper Zs value of surrander polygons for the current display window
ZNAX3	the maximum right-lower Zs value of surrander polygons for the current display window
ZNAX4	the maximum right-upper Zs value of surrander polygons for the current display window
ZNINMX	the maximum Zs value of the current polygon for the last display window
ZNIN1	the minimum left-lower Zs value of surrander polygons for the current display window
ZNIN2	the minimum left-upper Zs value of surrander polygons for the current display window
ZNIN3	the minimum right-lower Zs value of surrander polygons for the current display window
ZNIN4	the minimum right-upper Zs value of surrander polygons for the current display window
ZN	Zs value of the first vertex of the current edge
ZF	Zs value of the second vertex of the current edge

### 3. Hidden Surfaces removal and display symbols

EXLEFT	Xs value of left edge of the segment box
EXRGHT	Xs value of right edge of the segment box
EZLEFT	Zs value of left edge of the segment box
EZMAX	maximum Zs value of segment box
EZMIN	minimum Zs value of the segment box
EXRGHT	Zs value of right edge of the segment box
CHANGE	array used to mark polygons that have entering or exiting segments
CURSEG	index of current segment
DIV	division point of simple intersection of two segments within a span
DXLEFT	slope of Xs values of the left edge determining a segment
DXRGHT	slope of Xs values of the right edge determining a segment
DZLEFT	slope of Zs values of the left edge determining a segment
DZRGHT	slope of Zs values of the right edge determining a segment
EDGLST	index of the first edge linked in ENTLST
ENTLST	linked list of edges entering on a scan line
IACTIVE	array of the indices of the active segment blocks
IEFULL	used to indicate if a segment exceeds the x limits of the segment box
IBSEG1	index of the first segment of a simple intersection
IBSEG2	index of the second segment of a simple intersection
IBXCNT	number of segments in the box
IBXTYP	type of segment box
IFRELS	index of the next free segment block of storage

IMPLFT	used to flag that a span is bounded by an implied edge
IMPLST	index of current scan line sample point due to an implied edge
IMPLS2	index of first scan line sample point due to an implied edge
ISFULL	indicates if a segments Xs values exceed the span's Xs limits
IXSLFT	array containing the indices of the segment to left of any segment - an XSORT list
IXSRGT	array containing the indices of the segment to right of any segment - an XSORT list
IY	the scan line index
IYLEFT	array used to indicate on which scan line the left edge exits the display for each segment
IYENTR	array containing indices of the first edges which will enter on scan line IY
IYRGHT	array used to indicate on which scan line the right edge exits the display for each segment
LSTSEG	index of the last segment added to the display list for scan line IY
F	array containing the indices of the polygons common to each edge
PI	index of the current polygon
FOLCHG	index of first changing polygon on CHANGE list
POLGON	array containing the index of the polygon of each segment selected to be displayed
FOLSEG	array containing the indices of the first active segment for each polygon
PREVIS	index of the previous segment
SAMFRE	index of first free sample point storage location
SAMFST	index of the first sample point in list SAMX
SAMLNK	array of the indices for sample points which are ordered in a linked list

SAMLST	index of the last sample point added to the list
SAMPLE	index of the current scan line division or sample point
SAMX	array of the Xs values used to divide the current scan line
SDIV	the left-most endpoint of the current segment in a span
SEG	index of the current active segment
SEG1	index of new block of segment storage
SEGACT	index of the first segment of the list of segments which will intersect the next span
SEGCNT	number of segments to be displayed on scan line IY
SEGFST	index of first segment on the XSORT lists
SEGLO	index of last active segment examined
SEGLST	array used to link the list of a polygon's edges
SEGOUT	index of the first segment of the list of segments which intersect this span, but not the next
SXLEFT	Xs value of the left-most point of the current segment within the span
SXRGHT	Xs value of the right-most point of the current segment within the span
SZRGHT	Zs value of the right-most point of the current segment within the span
SZLEFT	Zs value of the left-most point of the current segment within the span
XLEFT	array of Xs values of the left edge of the active segments
XLSTSM	Xs value of last sample point
XRSL	the number of horizontal elements per scan line
XSPNLF	Xs value of the left edge of the current span
XSPNRG	Xs value of the right edge of the current

span  
XRIGHT array of Xs values of the right edge of the  
active segments  
ZRIGHT array of Zs values of the right edge of the  
active segments  
ZLEFT array of Zs values of the left edge of the  
active segments

## I. INTRODUCTION

Presentation of information using computer aided video graphical or pen plotting devices has become very important and extremely useful in almost every profession. Three-dimensional computer graphics is being utilized to display air combat training in the military, xray scans of the human body in medicine, and blue prints and stress characteristics for mechanical parts in industry. It has even been used to make animated movies. Since one manufacturer of graphical machines has not cornered the market, software and hardware standards have not been established. Each machine has a different screen area, smallest resolution size, and data structure. The lack of a standard graphics language becomes a source of expensive software re-writing every time a newer more capable graphics machine is purchased.

The existing video graphical devices can be divided into these four distinct categories: direct view storage tubes; vector generator cathode ray tubes (CRT's); raster scan CRT's; and plasma panels. The first three types of devices have achieved high resolution displays and are most commonly used. As a minimum, the hard copy devices have the capability of pen plotting and the more advanced machines produce shaded images using electrostatic plotting. Except for a plasma panel device, access to the above graphic machines was readily available at the Naval Postgraduate School. Although all of these devices were supported through Fortran IV, the software for one machine exhibited little discernible similarity to that for another. Each graphics device did have its own manual, but the actual

efficient utilization of an output terminal, at or near its designed capability, was typically left for experimental realization.

The experimental process, learning one machine's software idiosyncrasies, required an amount of time which was generally not reduced when learning those of a second. The high cost of software and this lengthy device learning process were the two main incentives urging the development of a graphics software package which could be used with any display device. Since Fortran IV has become a universally supported language, it was selected as the development software. Graphical presentations normally involve one of the following four types: two-dimensional (2-D) graphs; 2-D images; three-dimensional (3-D) graphs; or 3-D images. The first two types of presentations have been well documented and the theory for both has been fully developed. The theory for and usage of 3-D graphs has also received considerable attention. Programs for their display have been published in many software languages. Additionally, the Naval Station in Keyport, Washington was interested in a real time 3-D Torpedo presentation of the torpedo test area for the Range Safety Officer.

For these reasons, the scope of this research was limited to developing a three-dimensional (image) graphics package, written in Fortran, which would develop a data set that could be displayed on any selected device. A portable graphics package would reduce software costs for interfacing with a new device to a minimal effort. Usage of a 3-D graphics software throughout a large organization, like the Navy, would reduce a programmer's learning experience to a one time effort. An individual would then become a portable expert.

## II. THREE-DIMENSIONAL GRAPHICS

The proper display of a 3-D object on a 2-D surface, such as a video screen or a piece of paper, by a computer required that a complete numerical description of the object's boundaries or surfaces be supplied for processing. The two generally accepted methods used to represent 3-D objects are:

1. " surface definition using mathematical equations;
2. and surface approximation by planar polygonal mosaic." <sup>1</sup>

Either description required that a coordinate system be constructed to provide the numerical values.

### A. DISPLAY OF AN OBJECT AND ITS MOTION

A right-handed, cartesian coordinate system provided an acceptable and most generally understood system to describe 3-D objects. The unit of measurement was arbitrary with the only requirement that it was consistent. This system has been labeled the Object coordinate system. The complexity of the mathematics required to define the surfaces of an object with equations was only surpassed by the algorithms required to project the image onto a 2-D surface or alter the viewing aspect. It was much easier, both for understanding and utilization, to consider each object as a set of one or more polyhedra. Since a polyhedron consists of four or more planar sides, called polygons, the realistic approximation of non-planar

surfaces, such as a sphere, was determined by the number of polygons used. By definition, a polygon was a flat surface, a plane, bounded by three or more connected, straight lines, called edges. Two edges intersected at a single point, a vertex of the polygon.

Using the object coordinate system, a vertex was defined by its  $x$ ,  $y$ , and  $z$  values. With the origin placed at the geometric center of the object being described, the location of the vertices was simplified. By indexing each vertex as it was specified, an edge was determined by the indices of the two vertices which were its endpoints. Similarly, indexing each edge allowed the description of a polygon using the indices of the edges forming its boundary. Finally, a polyhedron was specified with the indices of the polygons which formed its planar surfaces.

With this type of object definition, object motion was implemented by moving its vertices in a linear manner. Linear movement of the vertices preserved the straightness of the polygonal edges and the structural purity of each polyhedron. Translation, which was the linear displacement of an object, was defined by the subtraction of a distance,  $T_x$ ,  $T_y$ , or  $T_z$ , from the respective vertex coordinate value,  $x$ ,  $y$ , or  $z$ . Thus, translation produced an entirely new set of vertex values, which could be stated as:

$$\begin{aligned}x' &= x - T_x \\y' &= y - T_y \\z' &= z - T_z\end{aligned}$$

Translation coupled with rotation of a point provided the ability to describe any 3-D motion. Rotation of a point  $P$  was most easily explained in two dimensions. Thus, in

Figure 1, P was rotated through the angle theta about the origin into the point P' by the transformation:

$$x' = x \cos \theta + y \sin \theta$$

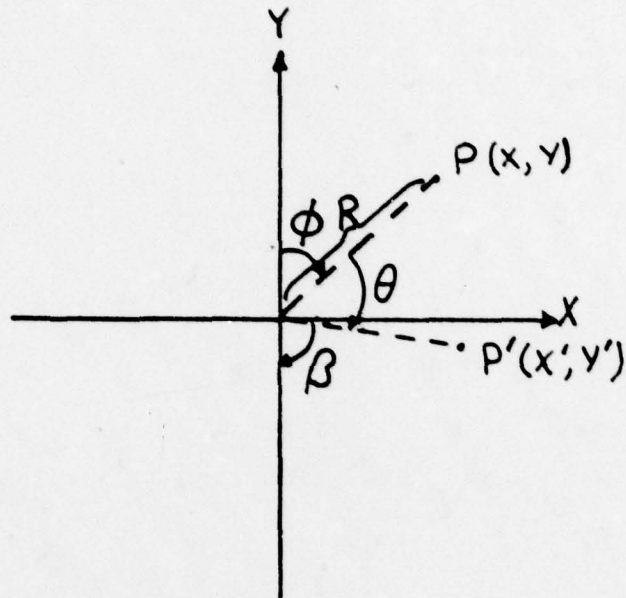
$$y' = x \sin \theta - y \cos \theta$$

This derivation for 2-D rotation, as provided in Figure 1, was directly applied to 3-D rotation about the Z-axis. The angle of rotation was measured in the clockwise direction looking from the positive infinity of the axis about which a point was to be rotated. Confining the rotation of points, and thus objects, to a combination of 2-D rotations greatly simplified the computer implementation.

The last transformation utilized to alter the viewing aspect of an object's image was scaling. This algorithm required that the vertex coordinate values be multiplied by the scale factors  $S_x$ ,  $S_y$ , and  $S_z$ . Provided the scale factors were of equal magnitude, the scaling was linear and preserved the polyhedron's shape.

To project a 3-D object onto a 2-D surface required two transformations. The first, called the viewing transformation, mapped the object coordinates into a system which had its origin at the viewpoint, or the eye, of the graphic software's user. This coordinate system preserved the object's linearity and produced the image of the object as seen by the "eye". Hence, it was called the Eye coordinate system. Its  $Z_e$ -axis was used to represent, or measure, the depth of the images. The system's  $X_e$  and  $Y_e$  axes were aligned with the horizontal and vertical dimensions of the display screen, respectively. As shown in Figure 2, the viewing transformation constructed a left-handed cartesian coordinate system.

The second transformation simply projected the eye coordinate points onto the plane of the display screen. This entire transformation was easily constructed and explained in Section III. E. 2., using Figure 12.



$$\theta + \phi + \beta = 180^\circ$$

$$\cos \beta = \cos (\pi - \phi - \theta)$$

$$\sin \beta = \sin (\pi - \phi - \theta)$$

$$Y'/R = -\cos (\phi + \theta)$$

$$X'/R = \sin (\phi + \theta)$$

$$= -\cos \phi \cos \theta + \sin \phi \sin \theta \quad = \cos \phi \sin \theta + \sin \phi \cos \theta$$

$$Y'/R = -Y/R \cos \theta + X/R \sin \theta \quad X'/R = Y/R \sin \theta + X/R \cos \theta$$

$$Y' = X \sin \theta - Y \cos \theta \quad X' = X \cos \theta + Y \sin \theta$$

Figure 1 - TWO-D ROTATION OF A POINT

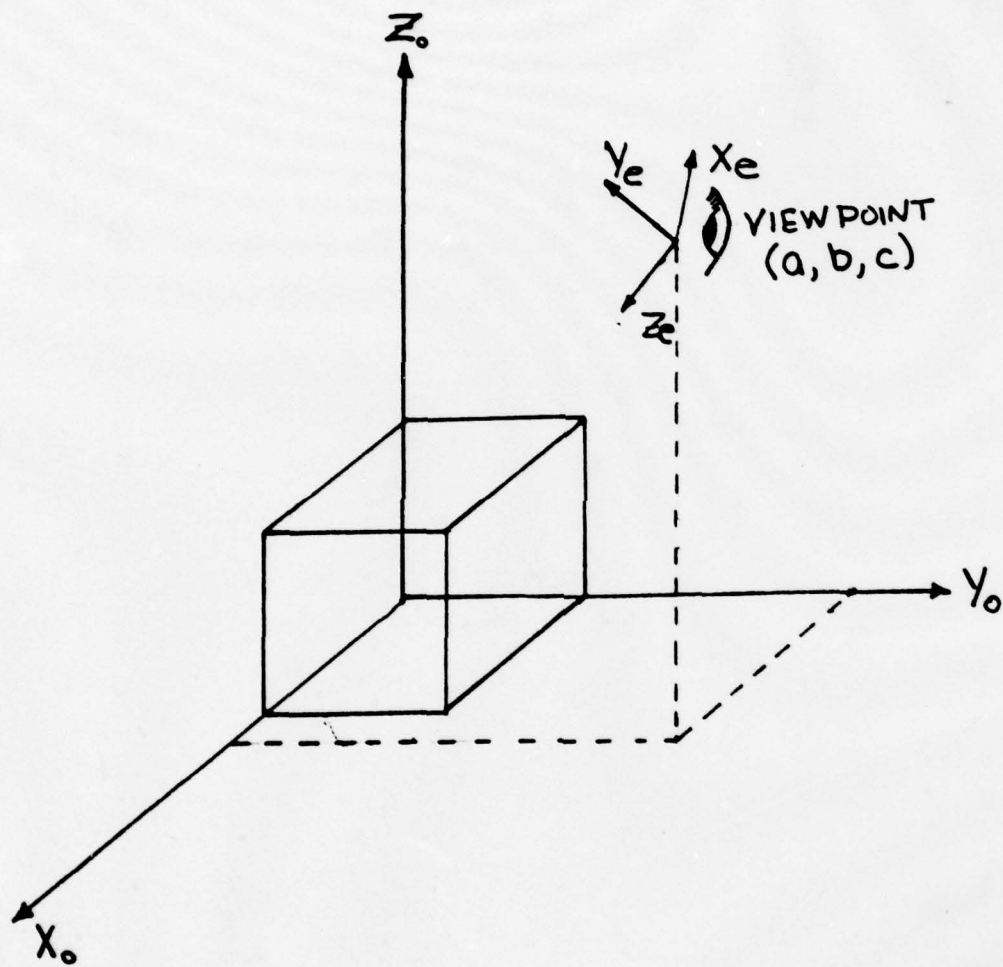


Figure 2 - EYE COORDINATE SYSTEM

## E. IMAGE REALISM

The perspective projection of an object onto a screen often produced points, and thus vectors, which could not be displayed. These vectors could be located behind the observer or simply off the screen. The first case would have caused erroneous vectors to be displayed, which could not actually be seen. Typically, the second would generate program failure, because the machine could not display points, or vectors located off the screen. Therefore, the non-viewable portion of any image had to be eliminated or cut away from the viewable section. This procedure was called image clipping.

Initial 3-D displays portrayed objects as wire-framed images. A polyhedron presented in Figure 3, was shown with all edges displayed. Because this was a simple body, the dedicated observer usually recognized it as a representation of a 3-D object. However, the correct viewing aspect (i.e. which surfaces were closest to the viewer) could not positively be ascertained. For this reason, the first effort to have a computer determine a more realistic presentation was the hidden line removal algorithm. Using one of these algorithms, the display shown in Figure 4 was drawn. The proper viewing aspect was instantly apparent provided the viewer recognized the display as a 3-D image.

The next computer graphics effort to increase display realism was to have the computer generate solid polygonal surfaces. This required that the display device had a shading or full color capability. For realistic images, the computer had to possess either a software or hardware implemented hidden surface removal algorithm. Now, the

viewable image's surfaces were displayed as shaded or colored polygonal planes with user supplied shading or colors.

Most recent research has concentrated on producing realistic, computer generated shading algorithms. Using the most complex shading procedures particular elements of image realism have all been achieved. However, a single universal solution to the proper shading of images has not been realized, due to different types of light sources, various material textures, and the lack of a uniform material reflectivity of light. To greatly decrease display processing time, most of the image realism programs were implemented in hardware after procedure refinement. Thus, computer graphics realism has become a function of software time available or of the cost of complex hardware.

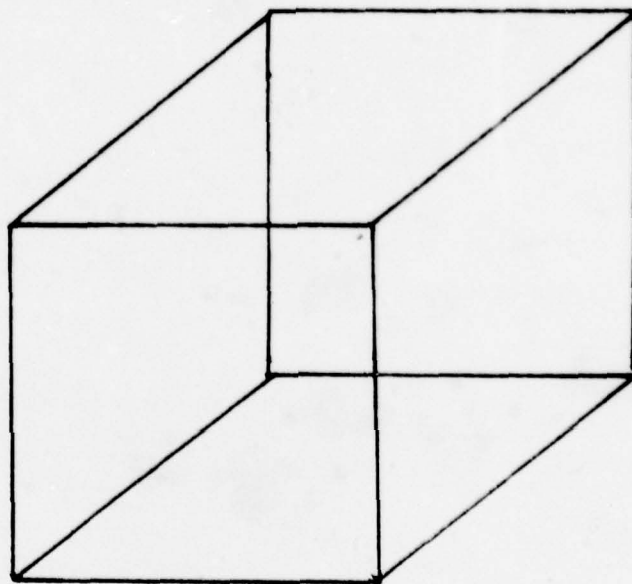


Figure 3 - WIRE-FRAME IMAGE

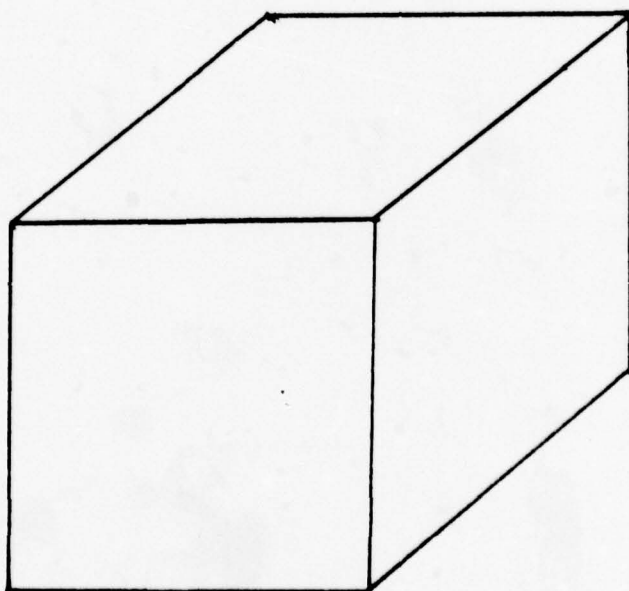


Figure 4 - HIDDEN LINES REMOVED

### C. GRAPHICS SOFTWARE STRUCTURE

In order to provide a "state-of-the art", computer graphics software package, the following capabilities were determined to be the minimal requirements:

1. Object to screen coordinate transformation;
2. Clipping algorithm;
3. Image scaling;
4. Image translation;
5. Image rotation;
6. Image shading;
7. Hidden line removal;
8. Hidden surface removal.

These capabilities enabled the viewing of an image from any aspect and the generation of realistic displays in "real time".

The data structure utilized was selected for its ease of user implementation and its applicability to the input requirements of the eight procedures above. To explain the capabilities above (in Section III.), each point, or vertex, was defined as a one by four vector, such as:

$[x \ y \ z \ w]$ , where  $w$  was a scale factor of the 3-D vector (normally  $w = 1$ ).

This type of vector representation allowed all transformations to be defined as a four by four matrix. While the complete logic for matrix and vector utilization was provided in the Appendices of Ref. [1], a key advantage was that it facilitated the concatenation of transformations. Concatenation of matrices can be simply explained with the following example.

$$\begin{aligned} [x' \ y' \ z' \ 1] &= [x \ y \ z \ 1] \underline{A} \\ [x'' \ y'' \ z'' \ 1] &= [x' \ y' \ z' \ 1] \underline{B} \end{aligned}$$

Can be equivalently stated as:

$$[x'' \ y'' \ z'' \ 1] = [x \ y \ z \ 1] \underline{T}, \quad \text{where } \underline{T} = \underline{A} \underline{B}$$

In the next section, the algorithms used to develop the graphics software package were briefly presented. Since every display device has a smallest horizontal and vertical resolution size, the proper construction of an image required the computational utilization of these machine dependent features. An easy means to visualize this concept for any device was to construct a  $n$  by  $m$  dot matrix, where  $n$  was the number of horizontal lines and  $m$  was the number of dots per line, as shown in Figure 5. Thus, to display an image required that line segments be drawn between the dots. Similarly, it was necessary to determine the screen's center in order to position the image in the middle of the display surface. The following terms were used consistently to define these quantities (see Figure 6):

- $V_{sx} = m/2$ , was half of the horizontal resolution size;
- $V_{sy} = n/2$ , was half of the vertical resolution size;
- $V_{cx} = m/2$ , was the screen's horizontal center;
- $V_{cy} = n/2$ , was the screen's vertical center.

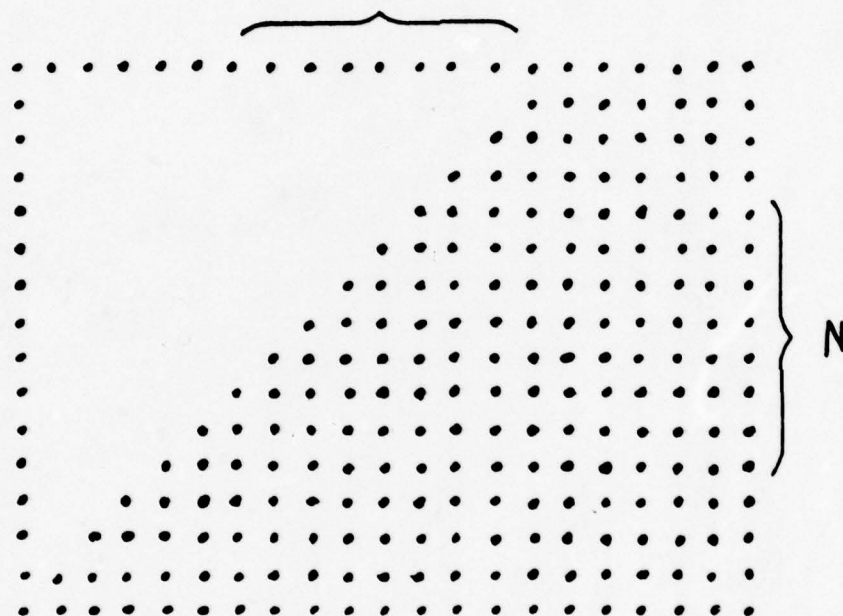


Figure 5 - N BY M DOT MATRIX SCREEN REPRESENTATION

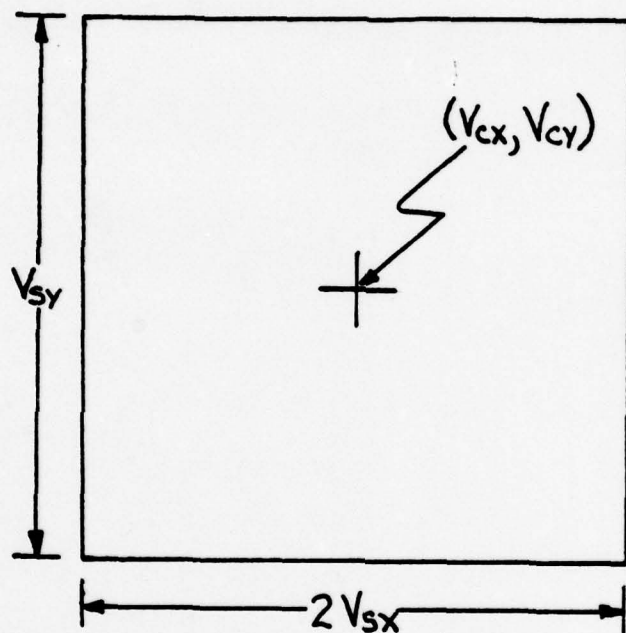


Figure 6 - DISPLAY SURFACE DIMENSIONS

### III. GRAPHICS SOFTWARE COMPONENTS

In this section the algorithm for each component was briefly presented along with any figures and general flow charts which aided with the explanation. For a detailed flow chart and Fortran program listing see Appendix A.

#### A. OBJECT COORDINATE SYSTEM -THE DATA BASE

The description of a 3-D object with this coordinate system allowed the user total flexibility in the selection of a convenient system of measurement and origin placement. Additionally, the user selected the viewpoint, which determined the viewing axis and thus, the initial viewing aspect. The viewing axis was the line defined by the viewpoint and the object system's origin.

Image definition started with the specification of its vertices. As each vertex was input to the computer, it was assigned a consecutive index number. Similarly, as the other sets of image elements (edges, polygons, and polyhedra) were input, they were consecutively indexed also. The x, y, and z object coordinate values were stored in three real arrays,  $XE(i)$ ,  $YE(i)$ , and  $ZE(i)$ . Integer arrays were used to store indices describing edges, polygons, and polyhedra. An edge was described by storing the index of one vertex in  $EDGE1(i)$  and the second in  $EDGE2(i)$ . A polygon was defined by storing the indices of the edges which composed its boundary in the array  $POLYGN(i,j)$ . The indices of these edges must be input so that:

$$\{ \text{EDGE1}(i) \text{ OR } \text{EDGE2}(i) \} = \{ \text{EDGE1}(i+1) \text{ OR } \text{EDGE2}(i+1) \}$$

The polygons which described a polyhedron were input consecutively to reduce storage requirements. Thus, a polyhedron was described by storing the index of the first and the last polygon in the array POLYHE(i,2). This data structure allowed excellent image flexibility, since any polyhedron could be rotated, scaled, or translated without altering the remaining display.

#### E. IMAGE SCALING

The scaling transformation multiplied the object x, y, and z values by the scale factors  $S_x$ ,  $S_y$ , and  $S_z$  respectively. If the scale factors used were not equal, image distortion resulted.

$$[X' \ Y' \ Z' \ 1] = [X \ Y \ Z \ 1] \underline{S}, \text{ where } \underline{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

#### C. TRANSLATION

The translation of an image, in object coordinates, represents the physical movement of a 3-D object in the x, y, or z directions by the amounts  $T_x$ ,  $T_y$ , or  $T_z$ , respectively (or any combination of the three). The transformation  $T$  can be stated in matrix form as:

$$\underline{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T_x & -T_y & -T_z & 1 \end{bmatrix}$$

#### D. IMAGE ROTATION

The rotation of a 3-D object was broken into four distinct categories. Each involved the rotation of the image through an angle, theta, about an axis in object coordinates. The following axes of rotation define the four categories:

##### 1. X-Axis Rotation

The image was rotated about the X-axis through the angle theta. The angle was measured in the clockwise direction about the origin, looking towards the origin from the positive X-infinity. The transformation,  $\underline{R}_x$ , was defined as:

$$\underline{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

##### 2. Y-Axis Rotation

The image was rotated through the angle theta about the Y-axis, where the angle was measured as stated above in

1. looking from positive Y-infinity. The transformation matrix  $\underline{R}_y$ , was defined as:

$$\underline{R}_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 3. Z-Axis Rotation

The image was rotated about the Z-axis through an angle theta, which was measured as above from positive Z-infinity. The transformation matrix  $\underline{R}_z$  was defined as:

$$\underline{R}_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 4. An Arbitrary Axis of Rotation

The rotation of an image about an arbitrary axis required that the set of transformations below be performed on the image's object coordinates. Except for  $\underline{R}_3$ , these transformations were necessary to move the arbitrary axis into an axis for which a rotation algorithm (1., 2. and 3. above) already existed.

The arbitrary axis was specified by any two distinct points on it,  $(x, y, z)$  and  $(x', y', z')$ . The direction cosines for the axis,  $a_x$ ,  $b_y$ , and  $c_z$ , were found by:

$$a_x = (x'-x)/R, \quad b_y = (y'-y)/R, \quad c_z = (z'-z)/R, \text{ where:}$$

$$R = \sqrt{(x'-x)^2 + (y'-y)^2 + (z'-z)^2}$$

One of the points,  $(x, y, z)$ , was translated to the origin using the transformation  $\underline{T}$ , where:

$$\underline{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x & -y & -z & 1 \end{bmatrix}$$

Now, the arbitrary axis was rotated into the Z-axis, by first rotating it about the X-axis through the angle alpha, as shown in Figure 7. This transformation,  $\underline{R}_1$ , which placed the arbitrary axis in the X-Z plane, was defined as:

$$\underline{R}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ where:}$$

$$\cos \alpha = c_z / v$$

$$\sin \alpha = -b_y / v$$

$$v = \sqrt{b^2 + c^2}$$

The arbitrary axis was then rotated about the Y-axis through an angle beta, as shown in Figure 8, into the Z-axis. This transformation matrix,  $\underline{R}_2$ , was described as:

$$\underline{R}_2 = \begin{bmatrix} v & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & v & 0 \\ x & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The rotation about the arbitrary axis through the angle theta was now defined as a rotation about the Z-axis. Thus, the transformation,  $\underline{R}_3$ , was exactly as described in 3. above for  $\underline{R}_2$ . The remaining task was to return the arbitrary axis, and the image, back to its original spacial position. This was accomplished by multiplying with the inverse of the transformations  $\underline{T}$ ,  $\underline{R}_1$ , and  $\underline{R}_2$  in the reverse order. Using the principle of concatenation, the total transformation may be stated as :

$$\underline{R}_T = \underline{T} \underline{R}_1 \underline{R}_2 \underline{R}_3 \underline{R}_2^{-1} \underline{R}_1^{-1} \underline{T}^{-1}$$

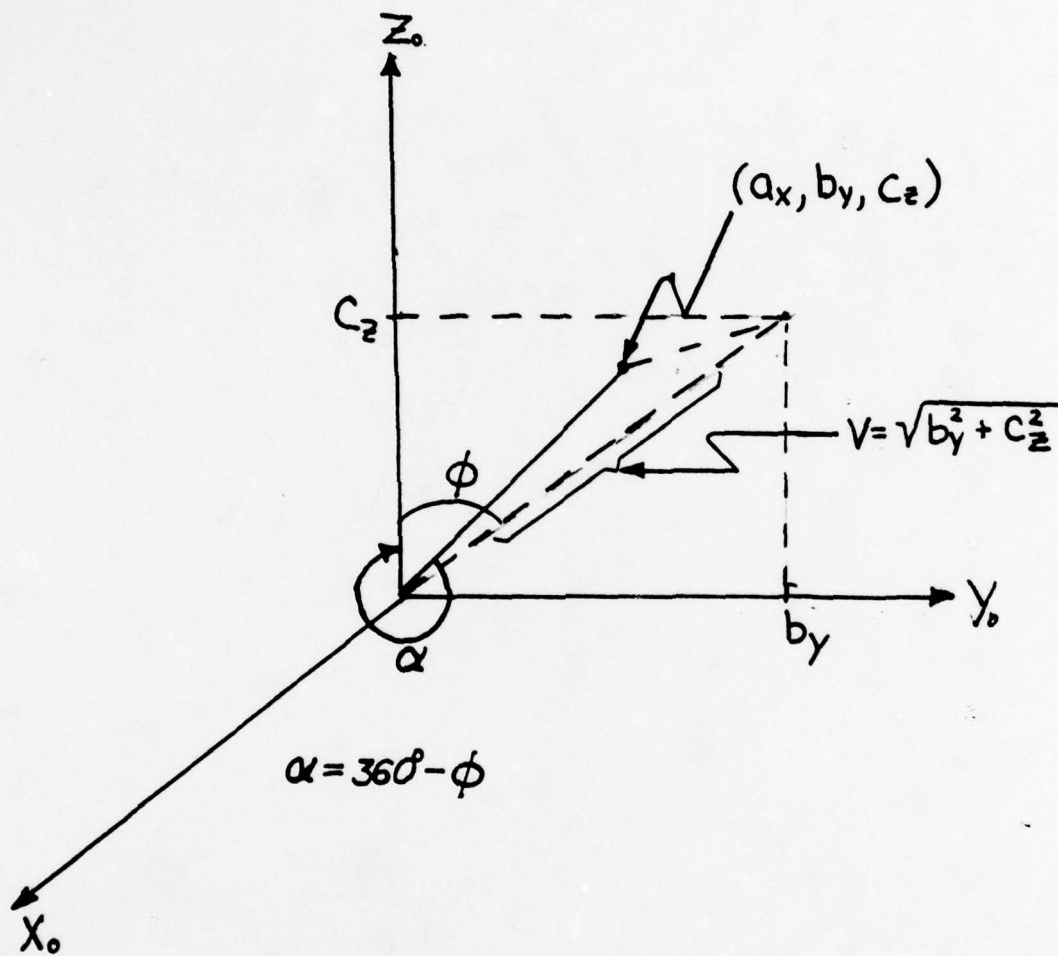


Figure 7 - ROTATION INTO X-Z PLANE

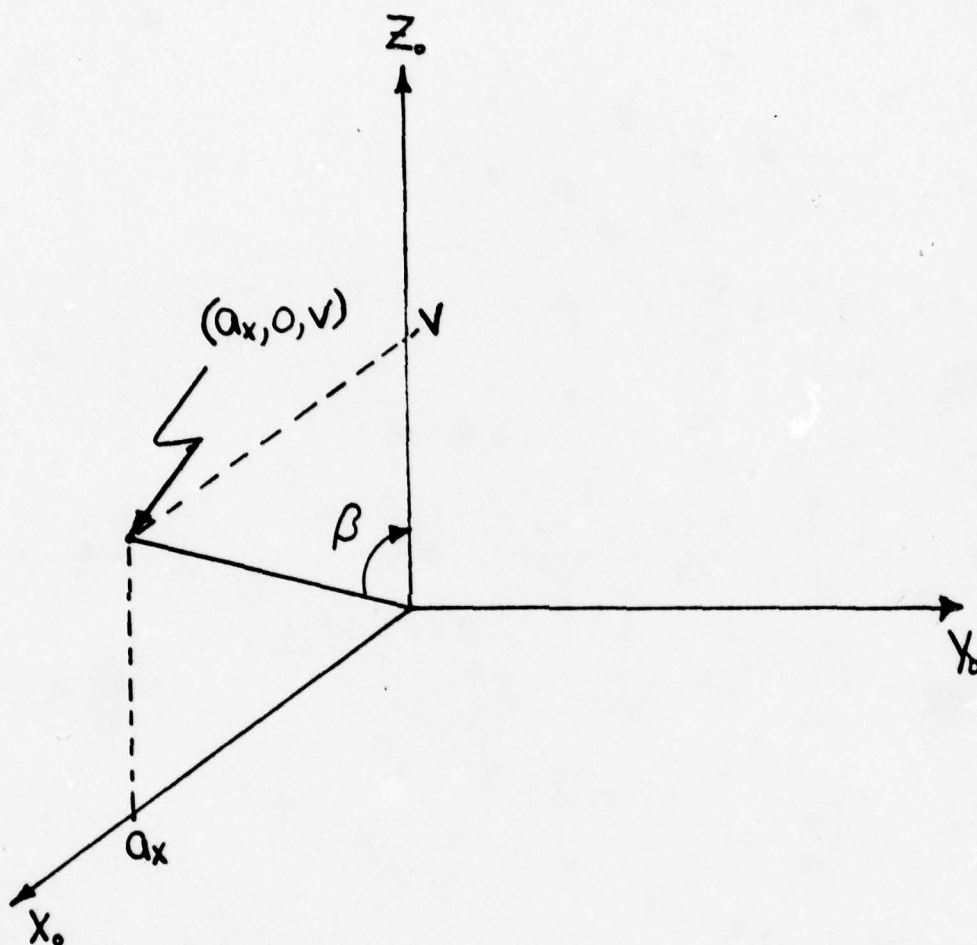


Figure 8 - ROTATION INTO Z-AXIS

## E. COORDINATE TRANSFORMATIONS

To display a 3-D object on a 2-D screen, a perspective projection was performed so that an object, such as a cube, when viewed orthographically (as on the screen) gave the proper perception of a 3-D form.

3

### 1. Object to Eye Coordinate Transformation

The left-handed, Eye coordinate system was utilized to determine the proper perspective view of any object. To eliminate the extremely complex viewing angle computations incurred by placing the observer close to the object (i.e. in the near field), the viewer was located an infinite distance from the object coordinate origin on the viewing axis. This allowed the rays emanating from an observer's eye to all be parallel to the viewing axis, the  $Z_e$ -axis, at the object. Parallel viewing rays allowed an orthographic projection of the 3-D object onto a 2-D screen with the  $Z_e$ -axis representing viewing depth. The  $X_e$  and  $Y_e$  axes were then aligned with the screen's horizontal and vertical dimensions, respectively. The eye coordinate system, as shown in Figure 9, preserved the linearity of the image. The transformation from the object to the eye system was called the viewing transformation,  $V$ , and was defined as:

$$\begin{bmatrix} X_e & Y_e & Z_e & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} V, \text{ where:}$$

$$V = T_1 T_2 T_3 T_4$$

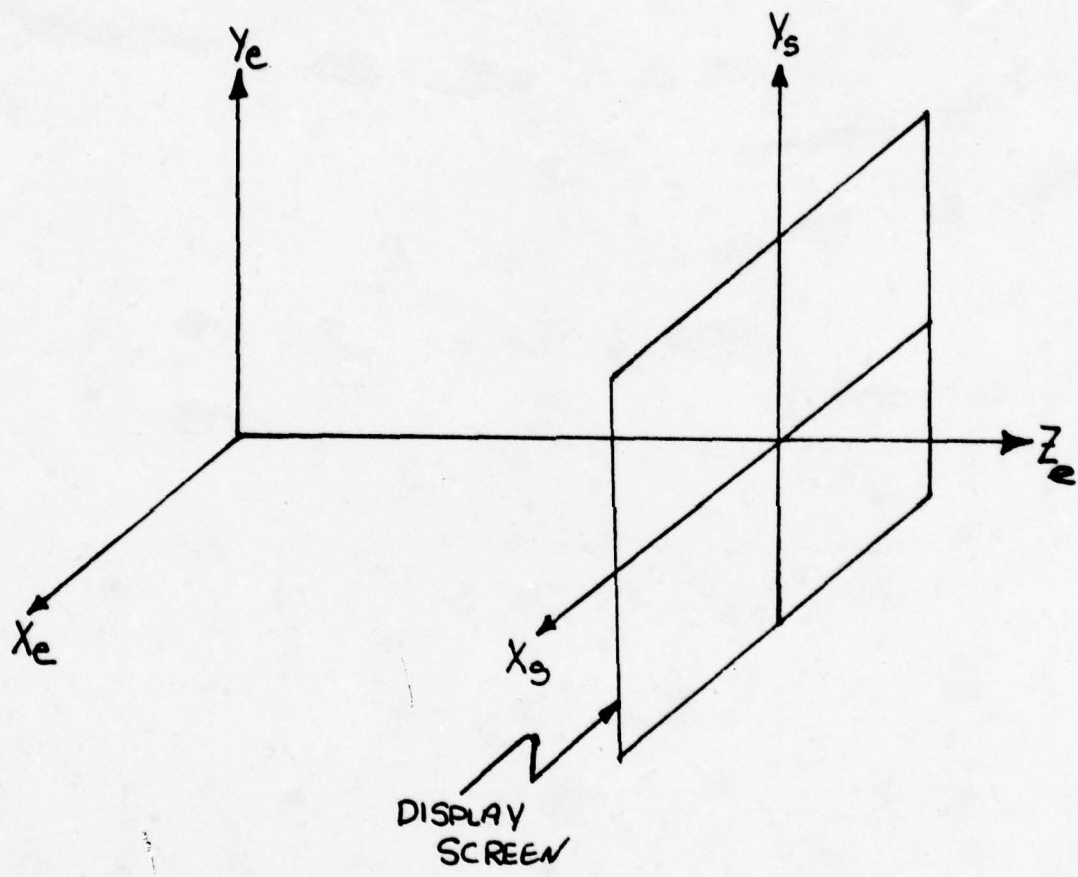


Figure 9 - EYE COORDINATE SYSTEM

Prior to this first coordinate transformation, the object coordinate values for the vertices were placed in the arrays XS(i), YS(i), and ZS(i). These arrays were used to generate the displayable information by the clipping, hidden line removal, or hidden surface removal algorithms. This prevented the original object's description from being destroyed or altered in these three procedures and allowed the graphics package to subsequently present different viewing aspects. Additionally, since clipping could remove an entire edge, the vertex indices were stored in EDGE(2,i) as:

$$\text{EDGE}(1,i) = \text{EDGE1}(i) \quad \text{and} \quad \text{EDGE}(2,i) = \text{EDGE2}(i)$$

The transformation matrices specifying  $V$  were formed as shown below when the viewpoint was located at (a,b,c) (in object coordinates) and the object was centered at the origin. Transformation  $T_1$  translated the viewpoint to the origin by :

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}$$

A left-handed cartesian coordinate system was formed with  $T_2$ , where:

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ c & 0 & 0 & 1 \end{bmatrix}$$

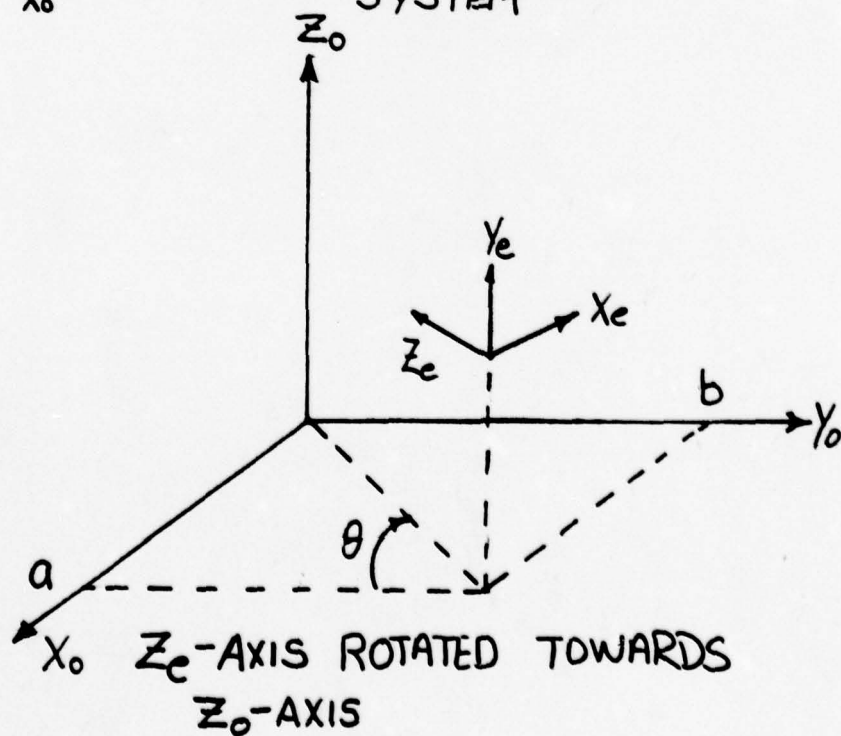
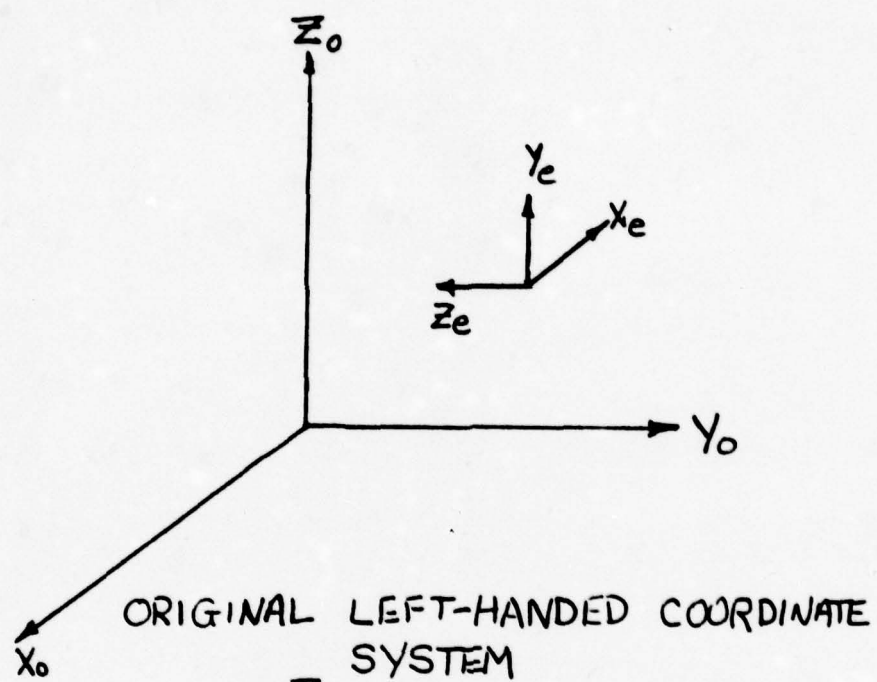


Figure 10 - VIEWING TRANSFORMATION

As shown in Figure 10, the system was rotated about the  $Y_e$ -axis through the angle theta which pointed the  $Z_e$ -axis at the point  $(0,0,c)$ . The transformation  $T_3$  was specified as :

$$T_3 = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ where:}$$

$$\cos \theta = a/v, \quad \sin \theta = b/v$$

$$v = \sqrt{a^2 + b^2}$$

Next, the coordinate system was rotated about the  $X_e$ -axis through the angle phi, as shown in Figure 11. This pointed the  $Z_e$ -axis towards the object space origin, where:

$$T_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ for:}$$

$$\cos \phi = v/(v^2+c^2), \quad \sin \phi = c/(v^2+c^2) \quad 4$$

This transformation arbitrarily selected the viewing axis as the line between the viewpoint and the object coordinate axis. It also placed the  $X_e$ -axis in the object system's  $Z = c$  plane. Since the object coordinate system was user defined the logical initial viewing aspect was to look at the system's origin or center. The initial view due

to the position of the  $X_e$ -axis was acceptable and basically irrelevant since the image could be rotated, translated, or scaled to provide any desired viewing aspect.

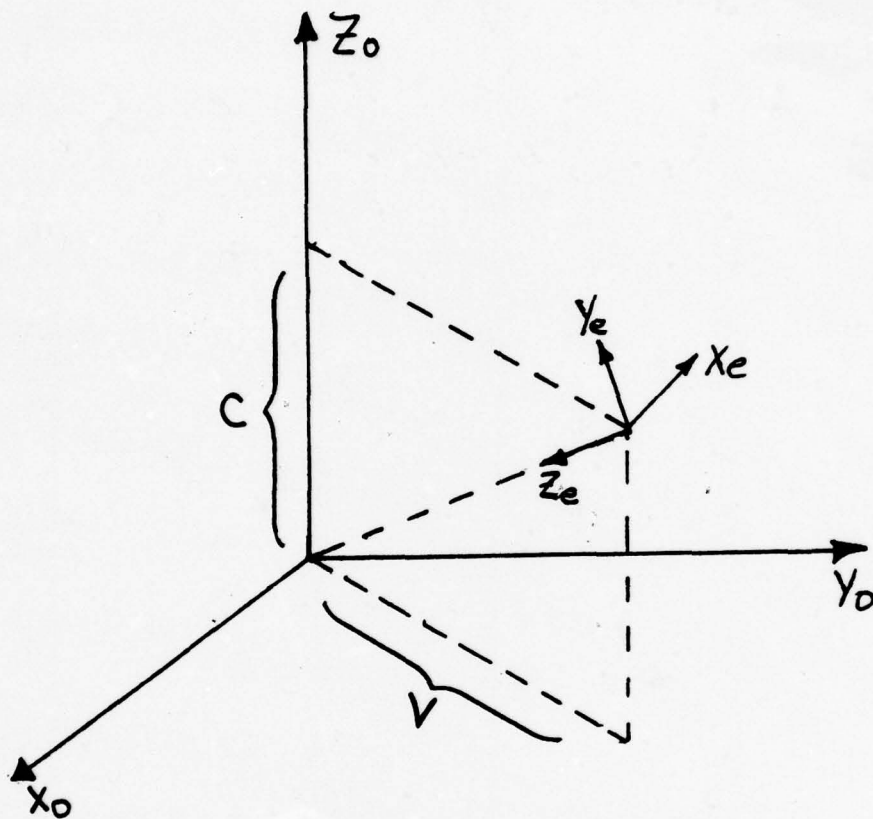


Figure 11 - Z -AXIS ROTATED TOWARDS ORIGIN  
e

## 2. Eye to Screen Coordinate Transformation

This transformation completed the perspective projection of the 3-D object onto the screen. As shown in Figure 12, the display was generated by simply projecting an object's eye coordinates onto the plane of the screen.<sup>5</sup> For a square display screen, one with equal horizontal and vertical resolution, the image was constructed without distortion, by the following transformation:

$$\begin{aligned} X_s &= S_x (X_e / Z_e) V_{sx} + V_{cx}, \text{ where } S_x = a/b \\ Y_s &= S_y (Y_e / Z_e) V_{sy} + V_{cy}, \text{ where } S_y = a/b \\ Z_s &= -1/Z_e \end{aligned}$$

If the viewing screen was not square then  $S_x$  and  $S_y$  were modified as shown:

$$\begin{aligned} \text{If } V_{sx} > V_{sy} \text{ then } S_x &= (a/b) (V_{sy} / V_{sx}) \\ \text{If } V_{sy} > V_{sx} \text{ then } S_y &= (a/b) (V_{sx} / V_{sy}) \end{aligned}$$

This modification was necessary so that an un-distorted image could be displayed over an entire rectangular screen. Without it one display dimension, typically the horizontal, would have been elongated.

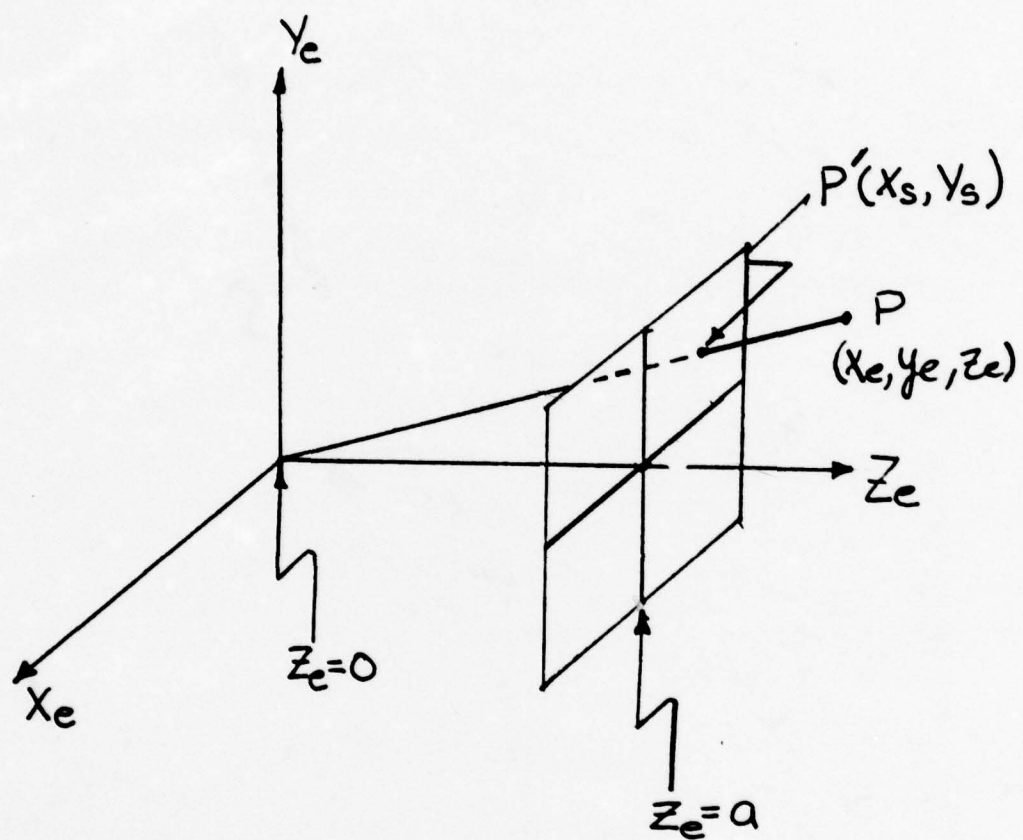


Figure 12 - PROJECTION OF DISPLAY POINTS ONTO SCREEN

## F. DISPLAY CLIPPING

The clipping procedure constructed a viewing pyramid which eliminated the undesirable effects of the perspective projection from object to screen coordinates, which were:

1. points and thus objects may have been located behind the viewpoint;
2. and objects may have exceeded the limits of the viewpoint (i.e. were located off the screen - were non-displayable).

The clipping of an image was performed on the image's data while it was expressed in eye coordinates to simplify the operation (as explained in Ref. [1]).

As shown in Figure 13, the geometry of the viewing pyramid dictated that for a point to be visible the following conditions must be satisfied:

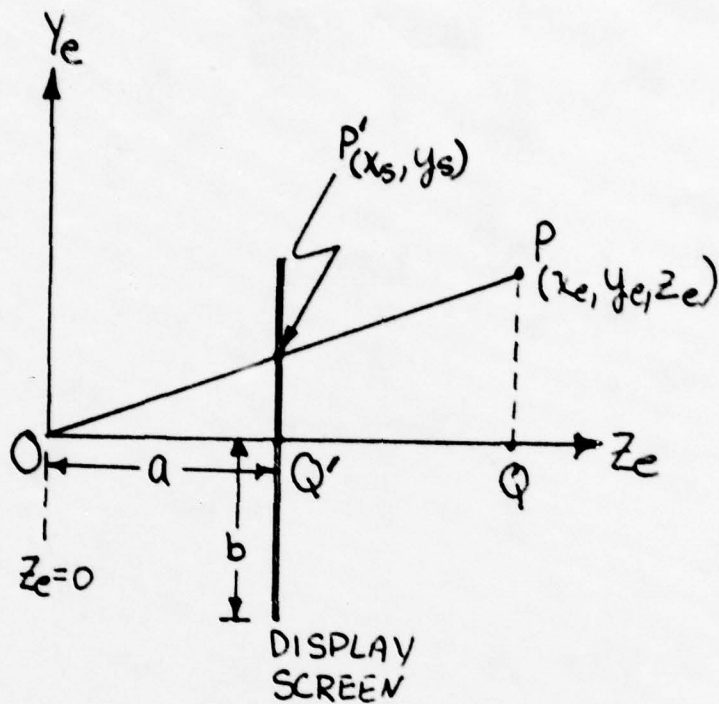
$$-Z_e \leq S_x X_e \leq Z_e \quad (1)$$

$$-Z_e \leq S_y Y_e \leq Z_e \quad (2)$$

Thus, a transformation from eye coordinates to a "clipping" coordinate system was described as:

$$\begin{bmatrix} X_c & Y_c & Z_c & 1 \end{bmatrix} = \begin{bmatrix} x_e & y_e & z_e & 1 \end{bmatrix} N, \quad \text{where:}$$

$$N = \begin{bmatrix} a/b & 0 & 0 & 0 \\ 0 & a/b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



IF  $(\frac{y_e}{b}) |Y_e| > z_e$  THEN POINT  $P'$  WILL  
BE OFF THE SCREEN

Figure 13 - CLIPPING COORDINATES

001	1000	1010
	DISPLAY SCREEN	
0001	0000	0010
0101	0100	0110

BIT 0: LEAST SIGNIFICANT BIT

BIT 0 = ICHK(I,1)

BIT 1 = ICHK(I,2)

BIT 2 = ICHK(I,3)

BIT 3 = ICHK(I,4)

BIT 3: MOST SIGNIFICANT BIT

Figure 14 - DISPLAY SCREEN DIVISION AND CODING

This coordinate system was established to display only viewable points. Additionally, if an edge's endpoint was located outside of the viewing pyramid, this routine located a point on the edge which satisfied equations (1) and (2) above and became the new endpoint. By dividing the plane of the screen, the  $X_s - Y_s$  plane, into nine sectors (Figure 14), the location of the two vertices of an edge was determined using the inequalities (1) and (2).

Since a vertex can be used as an endpoint of several edges, the clipping procedure placed the  $x$ ,  $y$ , and  $z$  values of the two endpoints into the arrays  $X(2)$ ,  $Y(2)$ , and  $Z(2)$ , respectively, as each edge was examined. Thus, if new endpoints had to be computed for this edge, the values of the original vertices were not destroyed. Since Fortran IV did not support binary operations, an integer array,  $ICLK(2,4)$ , was used to code the location of each endpoint as follows:

```

If  $X(i) < -Z(i)$  then  $ICLK(i,1) = 1$  else  $ICLK(i,1) = 0$ 
If  $X(i) > Z(i)$  then  $ICLK(i,1) = 1$  else  $ICLK(i,1) = 0$ 
If  $Y(i) < -Z(i)$  then  $ICLK(i,3) = 1$  else  $ICLK(i,3) = 0$ 
If  $Y(i) > Z(i)$  then  $ICLK(i,4) = 1$  else  $ICLK(i,4) = 0$ 

```

If both endpoints were displayable, no action was taken and the next edge was examined. If both vertices were to the right of the viewing pyramid (or both to the left, or both above, or both below), the entire edge was deleted from the display (i.e. if  $ICLK(1,j) = ICLK(2,j) = 1$  then discard the edge). Until both endpoints were displayable or the edge could be rejected, new points had to be computed on the edge. This computation has often been termed "pushing" the endpoint towards the display area. The pushing of the endpoint was accomplished by utilizing the 3-D, parametric

representation of a line to select that point where:

$$|X_c| = |Z_c| \text{ if inequality (1) was not satisfied;}$$

OR

$$|Y_c| = |Z_c| \text{ if inequality (2) was not satisfied.}$$

If inequality (1) was violated, the following sets of equations are used to compute the new endpoint:

If  $X(1) < -Z(1)$ : then:

$$t = [Z(1) + X(1)] / [(X(1) - X(2)) - (Z(2) - Z(1))]$$

$$Z(1) = t * [Z(2) - Z(1)] + Z(1)$$

$$X(1) = -Z(1)$$

$$Y(1) = t * [Y(2) - Y(1)] + Y(1)$$

If  $X(1) > Z(1)$  then:

$$t = [Z(1) - X(1)] / [(X(2) - X(1)) - (Z(2) - Z(1))]$$

$$Z(1) = t * [Z(2) - Z(1)] + Z(1)$$

$$X(1) = Z(1)$$

$$Y(1) = t * [Y(2) - Y(1)] + Y(1)$$

When inequality (2) was violated, the equations used to compute the new point were those above with every X replaced with a Y and vice versus.

#### G. HIDDEN LINE REMOVAL

The method utilized to remove hidden lines from 3-D objects was developed by John Warnock at the University of Utah. The program was interpreted from a SAIL program listed in Ref.[1]. This procedure required that the object coordinates be transformed to eye and then to screen coordinates without any intervening clipping of the image. The algorithm was broken into three main sections, the

Looker, the Thinker, and the Controller. The storage of the vertex indices in the array `EDGE(2,j)` was re-structured so that the index of a polygon's first edge could be used to link to the index of its second, and its second could link to the third edge, etc. Thus, an initializing subroutine linked each polygon's edges in the array `EDLINK(i)`.

The concept of linked lists uses the index of edge `i` to produce the index of the next edge for the same polygon. Since an edge could be common to two polygons, the first polygon to link the edge `i` found the storage location `EDLINK(i)` unused. This first polygon stored the index of its next edge, `j`, in `EDLINK(i)`. Because two polygons can have at most one edge in common, the second polygon to link edge `i`, could not use this same storage location, `EDLINK(i)`. Therefore, the number of edges, called `EDGEN`, was increased by one and the vertices for edge `i` were also stored as shown:

```
EDGE(1,EDGEN) = EDGE(1,i)
EDGE(2,EDGEN) = EDGE(2,i)
```

Additionally, the storage location `EDLINK(EDGEN)` was used to store the index of the next edge for this second polygon. This ordering usually doubled the storage requirements for edge definition. The vertex indices were further ordered in the array `EDGE(i,j)`, so that:

```
EDGE(2,j) = EDGE(1,j+1)
```

The data was structured, using linked lists (integer arrays) and pointers to the first element of the list, as follows:

1. `POLINK`-a list of polygon indices ordered by the polygon closest to the viewer with pointer `POLPTR`;
2. `POLEDG`- contained the indices of the first edge of

each polygon. The pointer used was the polygons index;

3. EDLINK-contained the linked list of edges. The pointer to a polygons second edge was the index of its first (found in PCLEDG);
4. POLLST-a list of polygons which were determined by the Locker to be either surrounders or intersector (which are explained below). The pointers to the list were SURRND and INTER.

An example of the usage of these linked lists was provided below:

INTER was the last polygon (index) added to the list of intersector and thus, was at the head of the list.

POLLST(INTER) = P, where P was the index of the second polygon on the intersector list.

PCLEDG(P) =  $E_1$ , where  $E_1$  was the index of the first edge for polygon P.

EDLINK( $E_1$ ) =  $E_2$ , which was the index of the second edge for polygon P.

EDGE(1,  $E_1$ ) =  $V_1$  and EDGE(2,  $E_1$ ) =  $V_2$ , where  $V_1$  and  $V_2$  were the indices of the two vertices describing edge  $E_1$ .

The concept of linked lists was utilized extensively for both the hidden line and hidden surface removal routines. In this procedure, a display window, which was initially the entire screen, was examined against each polygon. Each window could be classified as:

1. nothing was contained in this window;
2. the information contained in this window was simple and could be displayed;

3. or the information contained in the window was too complex.

Situations 1. and 2. resulted in a successful processing of the window. The next window on the stack could then be examined. The last classification was a failure and caused the window to be divided into four windows of equal size. These new windows were then pushed onto the stack.

The first important element needed to process a display window was the computation of the depth of the plane determined by a polygon at the four corners of the window. The planar equation can be stated as:

$$Ax + By + Cz + D = 0$$

The coefficients, A, B, C, and D, can be found from the x, y, and z values of any three points contained in the plane which are not colinear. Since the corners of a window are specified as  $X_s$  and  $Y_s$  values in screen coordinates, the depth of the polygon was computed by simple substitution into the plane equation.

The last important concept needed to process a window was the classification of each polygon as:

1. an intersector of the window;
2. a surrounder of the window;
3. or disjoint from the window.

These concepts are clearly portrayed in Figure 15. The classification of all polygons was performed by the Locker.

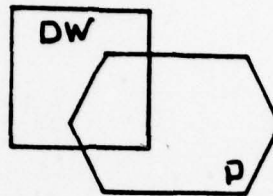
To determine whether a polygon was an intersector, it was sufficient to find any one of the polygon's edges which intersected the window. This determination was made by a clipping subroutine, which was very similar to that in P. If none of the edges intersected the window, the polygon was determined to be a surrounder or disjoint from the window by computing the angle "about the window through which each

edge passed." <sup>6</sup> The sum of these angles, as all edges of a polygon were processed, would equal  $\pm 360$  degrees, if the polygon was a surrounder of this window, as shown in Figure 16. The actual computation of each edge's angle was implemented by dividing the  $X_s - Y_s$  plane into nine sectors, as shown in Figure 17. The window was located in the center region, and the outer eight regions were numbered as shown. The endpoints were located, just as in the clipping routine in F. , and assigned the proper sector number.

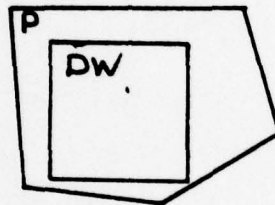
The edge's "angle" was the number of sectors which an edge entered, not counting the sector of the first endpoint. A polygon which surrounded the window had an "angle" of  $\pm 8$ , and a disjoint polygon had a zero angle. Extremely complex polygons could have an angle equal to  $\pm 16$ , or higher multiples of eight, by surrounding the window two or more times. However, usage of such complex polygons was unnecessary to construct any image. Because the incremental angle ( ) was defined as the difference between the sector values of the two endpoints, one problem of computing an edge's angle occurred when the magnitude was greater than four. Since no linear edge could enter more than four sectors (starting from the sector of the first endpoint), the edge's angle was adjusted when the magnitude was greater than four by :

If  $\Delta\alpha > 4$  then  $\Delta\alpha = \Delta\alpha - 8$   
 If  $\Delta\alpha < -4$  then  $\Delta\alpha = \Delta\alpha + 8$

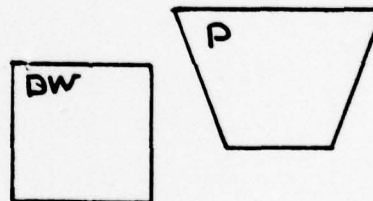
DW = DISPLAY WINDOW  
P = POLYGON



INTERSECTOR

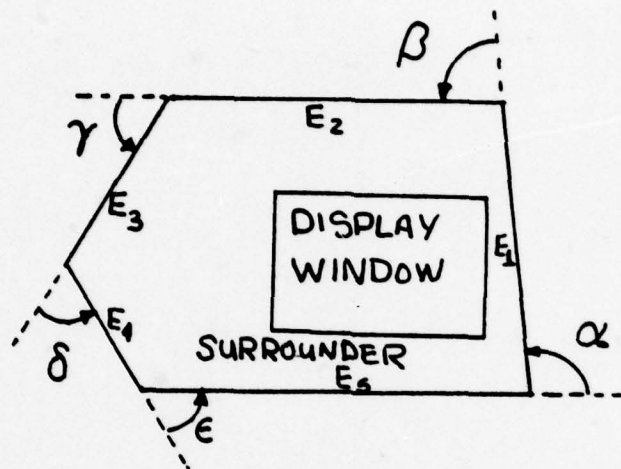


SURROUNDER



DISJOINT

Figure 15 - POLYGONAL CLASSIFICATIONS



$$\alpha + \beta + \gamma + \delta + \epsilon = 360^\circ$$

Figure 16 - EDGES'S ANGULAR COMPUTATION

3	2	1
4	DISPLAY WINDOW	0
5	6	7

Figure 17 - DISPLAY AREA DIVISION AND CODING

The last angle computational problem occurred when the magnitude was equal to four, as shown in Figure 18. The correct sign in this situation was not rendered by taking the difference of the two sector values. Counter-clockwise rotation or movement about a polygon's edges should have yielded a positive result. The problem was resolved by selecting any point between the two endpoints which was not in either of the vertices' sectors, as shown in Figure 19. By dividing the edge at this point, the correct angle could be computed by summing the angles of these "two edges". This angular computation for an edge was determined in the clipping subroutine used by this procedure.

The linked lists of the classified polygons were then passed to the Thinker. The surrounder list was processed first to determine which polygon was closest to the viewer by computing the depth of each of these polygons at the four window corners, as shown in Figure 20. Provided the closest polygon, called the hider, was not penetrated by another polygon, these four depths were used to determine if an intersector polygon was located completely in front of the hider within the confines of the display window. If an intersector was completely hidden from the viewer by the hider, it was removed from the list. If the final intersector list contained only one polygon, then that part of the polygon's edges which were inside the window were displayed. If the list contained more than one intersector, or if any intersector polygon penetrated the plane of the hider, the Thinker announced failure for that window. If the hider was penetrated by another surrounder, the Thinker announced failure before examining the intersector list. The penetration of one polygon by another was shown in Figure 21. Whether polygon B would be classified as an intersector or a surrounder penetrating polygon A would depend on the placement of the window. When the complexity

of the display could not be resolved and the size of the window had been reduced to the display device's smallest resolution, a dot was displayed at the window's lower, left corner. In this manner, the penetration of one polygon by another, which described a line, was displayed as an implied edge.

If failure was announced, and the size of the window was larger than the smallest resolution, the display window was divided into four equal windows and pushed onto the top of the stack. The Controller then selected the next window on the stack and passed it to the Locker. This entire process was summarized by the flow chart in Figure 22.

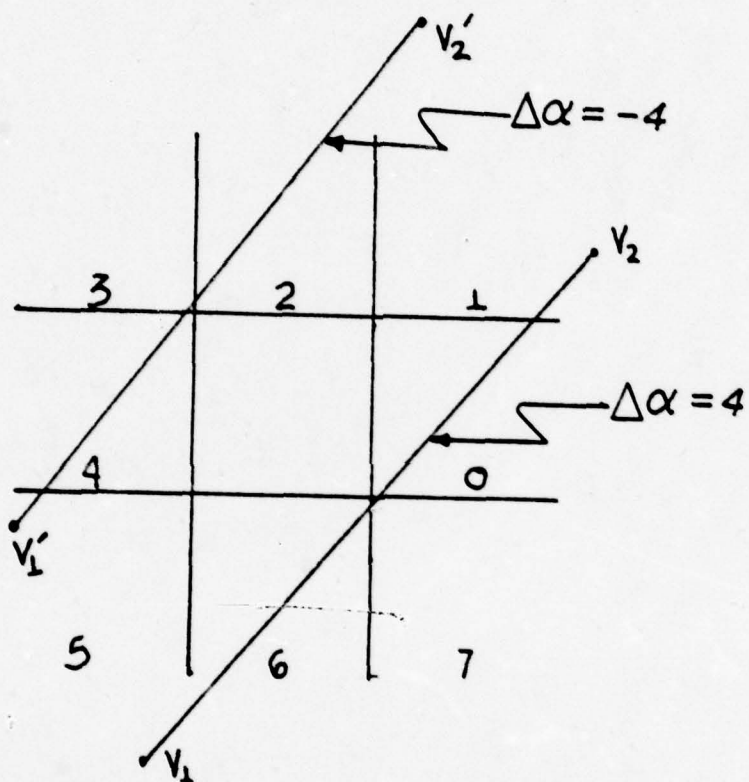


Figure 18 - ANGLE MAGNITUDE OF FOUR

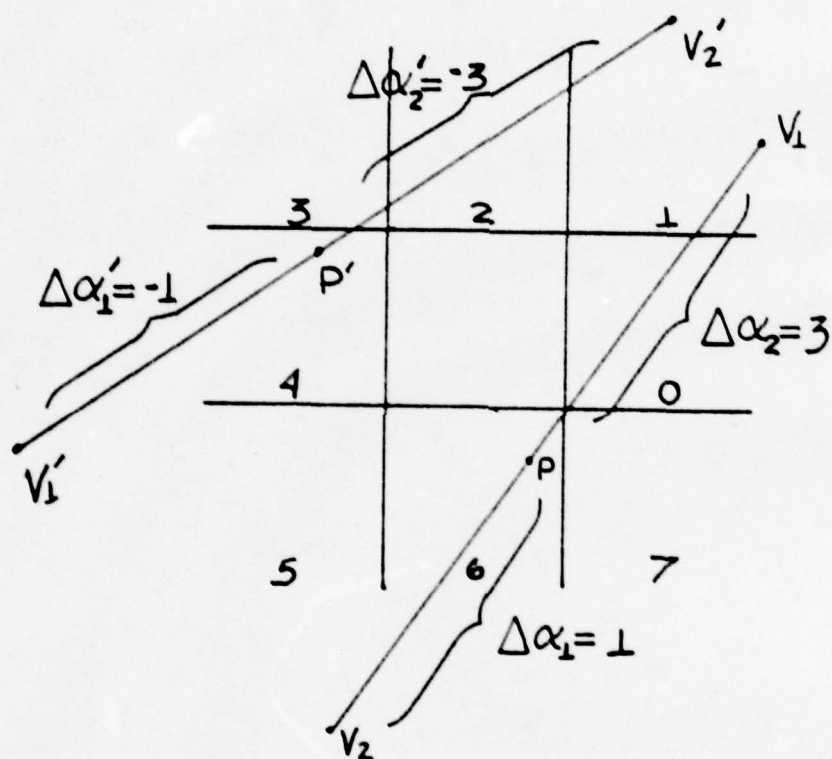


Figure 19 - EDGE DIVISION FOR PROPER ANGLE COMPUTATION

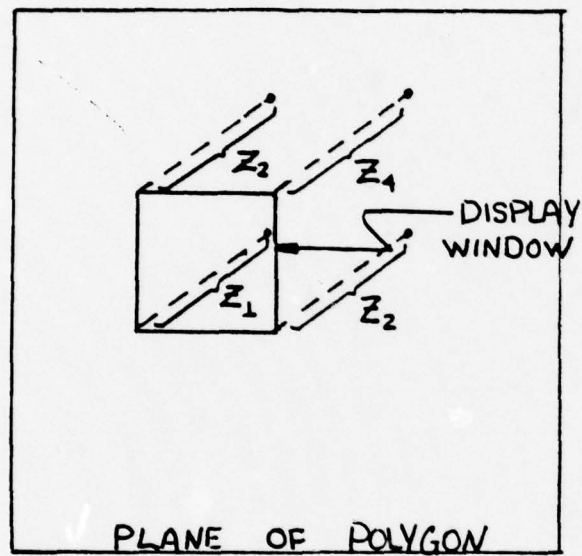


Figure 20 - DEPTH COMPUTATION OF POLYGON AT FOUR CORNERS OF WINDOW

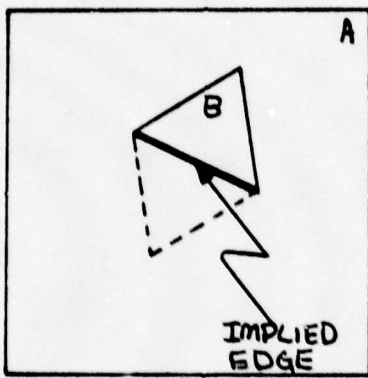


Figure 21 - POLYGONAL PENETRATION

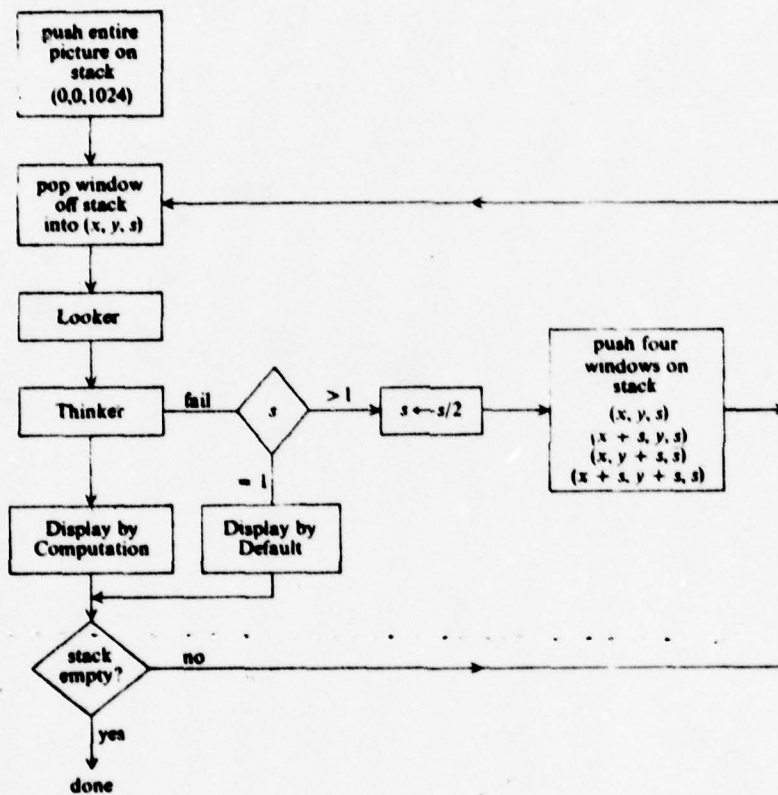


Figure 22 - HIDDEN LINE REMOVAL FLOW CHART

## H. HIDDEN SURFACE REMOVAL

The algorithm utilized for this procedure was developed by G. S. Watkins at the University of Utah. The program was interpreted from a SAIL program presented in Ref. [1]. While the hidden line removal algorithm concentrated on linked lists of polygons, the hidden surface algorithm processed the display with lists of edges. Additionally, the vertices had to be expressed in clipped, screen coordinates. Although the user still generated the data for an image as stated in A., this procedure displayed the polygonal surfaces as a solid plane using shading or colors. Thus, a 3-D object should become much more realistic when displayed with shaded surfaces vice wire frames. While the algorithms presented previously in section III., can be utilized on any display device, this concept was developed specifically for raster scan CRT's.

A raster scan is a special type of CRT, which is very similar to the television in most homes. The vectored CRT's and the direct view storage tubes generate a display by pointing an electron beam to a desired location on the display screen and then moving it to any other screen location. This process illuminates the phosphorous screen to produce a single line segment. The typical television receives an analogue broadcast signal which generates a single horizontal line of the screen's image at a time. At the end of each line a horizontal sync pulse is received to move the electron beam down one line, and to the left-hand edge. When the last horizontal line has been displayed a vertical sync pulse moves the beam to the top, left-hand corner of the screen. Since the phosphorous screen remains illuminated a very short time, the image must be constantly

refreshed, typically at a rate of thirty times per second.

A raster scan display device receives its image (and refresh) information from random access memory (RAM) refresh planes where the image is stored as a sequence of individual bits. Each bit of a memory plane determines the illumination of a single element on one horizontal display line (also called a scan line or a raster). A picture element, called a pixel, is the smallest screen resolution size. The standard sixteen level grey shading requires four bits, one bit on four planes, to represent the shading of one pixel. Similar memory requirements are needed to display an image with sixteen colors. While the vectored CRT's have achieved resolutions on a display screen of 4096 lines with 4096 elements per line, the finest resolution available with raster scan devices is 1024 by 1024. Thus, a sixteen color, raster scan display with high resolution required four million bits of RAM. Because of this extensive memory requirement, the development of this type of display device followed that of the small, lower cost electronic memory.

As shown in Figure 23, the intersection of the plane of a scan line with a polygon was a line segment. (Scan line  $k$  corresponds to the  $Y_s = k$  plane.) This line segment's endpoints were defined by its  $X_{left}$ ,  $Z_{left}$ ,  $X_{right}$ , and  $Z_{right}$  values, which were the  $X_s$  and  $Z_s$  coordinates of the intersection of the scan line with two of the polygon's edges. The two, 2-D equations below were used to find the intersection of an edge with each scan line.

$$X_s = a Y_s + b \quad (1)$$

$$Z_s = c Y_s + d \quad (2)$$

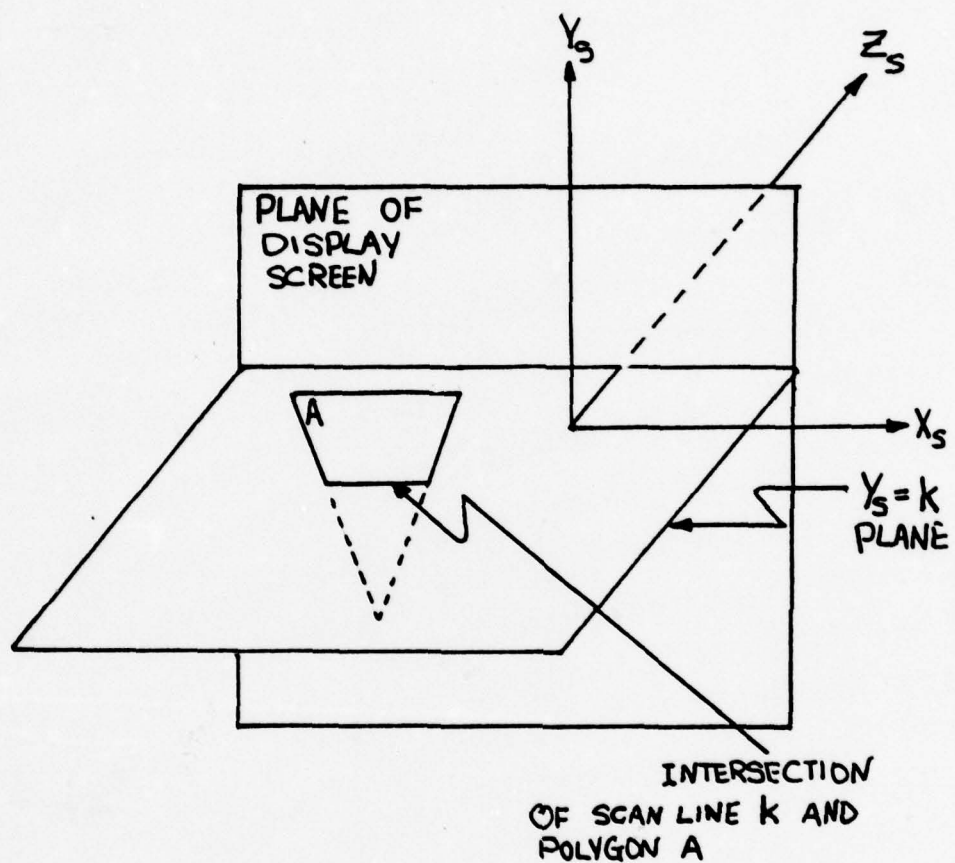
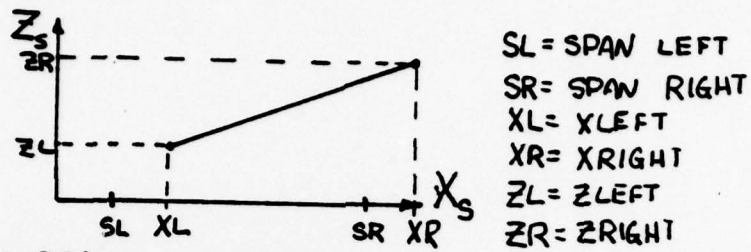
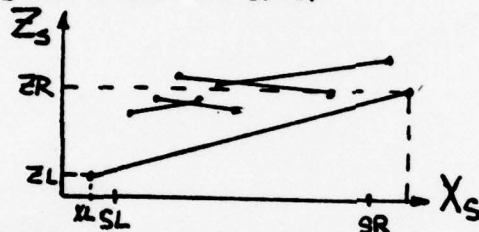


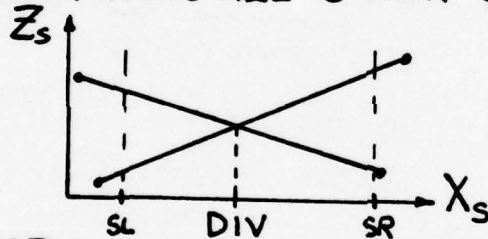
Figure 23 - SCAN LINE INTERSECTION OF A POLYGON



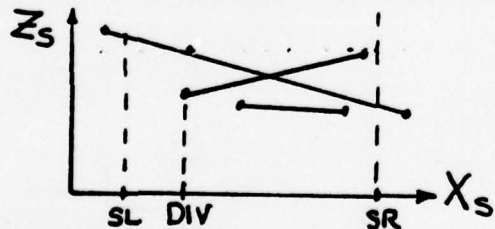
1. ONE SEGMENT IN SPAN



2. SPANNER HIDES ALL OTHER SEGMENTS



3. SIMPLE INTERSECTION OF TWO SPANNERS



4. TOO COMPLICATED - DIVIDE SPAN AT DIV

Figure 24 - SEGMENT CLASSIFICATION

The coefficients,  $a$ ,  $b$ ,  $c$ , and  $d$ , were quickly obtained using the equations for a 2-D line with the coordinates of the two vertices describing each edge.

With these equations the  $X$  and  $Z$  values of the intersection of an edge with scan line  $k+1$  was the values at scan line  $k$  plus their respective slopes,  $a$  and  $c$ . Since each edge, and thus each segment, sustained only an incremental change between scan lines, the display was also assumed to remain constant between scan lines. This scan line coherence of the display was used to decrease the time required to process an image.

To process each scan line, it was necessary to divide the line into spans which could be more easily resolved. The content of a span could be categorized as shown in Figure 24 and as described below:

1. "The span contained only one segment.
2. One segment was closer to the viewer than all others and it was a spanner. A spanner was a segment where  $X_{left} \leq \text{Span left}$  and  $X_{right} \geq \text{Span right}$ , as shown in Figure 24.
3. There was a simple intersection of the only two segments in the span, and both were spanners. This span was divided at the intersection into two spans and processed as in 1.
4. The display was too complicated in this span so it was divided at the left-most segment endpoint, or at the span's mid-point if there was no endpoint. The new spans were then processed."

7

Since the lower, left-hand corner of the raster scan machine at the Naval Postgraduate School, the RAMTEK, was indexed as  $(0,0)$  each edge was ordered so that the index of

the vertex with the largest  $Y_s$  value was stored in `EDGE(1,i)`. The integer value formed by truncating this vertice's  $Y_s$  value determined the first scan line that an edge would enter the display. A linked list of the indices of the edges which entered on each scan line was stored in the array `ENLIST(i)`. The index of the first edge to enter on scan line  $k$  was stored in `YENTER(k)`.

As each edge entered the display, the  $X$  and  $Z$  values and the scan line coherence factors, the slopes for equations (1) and (2), were computed. Since the object of this algorithm was to display polygonal surfaces, the indices of the current segments of a polygon were linked in `POLSEG(i)`. The segments were ordered by increasing  $X$  values of their left endpoint and the first segment's index was placed in `SEGLST(p)` for polygon  $p$ . A segment's index pointed to a block of storage which defined the endpoints  $X$  and  $Z$  values in the arrays `XLEFT(i)`, `ZLEFT(i)`, `XRIGHT(i)`, and `ZRIGHT(i)` and their respective slopes `DXLEFT(i)`, `DZLEFT(i)`, `DXRGHT(i)`, and `DZRGHT(i)`. Integer arrays `IYLEFT(i)` and `IYRGHT(i)` were used to indicate when an edge, the source of one endpoint of a segment, was exiting the display.

To properly insert an entering edge into a polygon's segment lists required the comparison of the edge's  $X$  value for this scan line to the `Xleft` and `Xright` values of all of the active segments. If two edges entered on the same scan line at the same  $X_s$  coordinate, as shown in Figure 25, the edge with the largest slope was entered first. Thus, to enter the two edges on scan line  $k+1$ , between the two existing edges, a new block of storage was added for each entering edge. As each edge exited the scene, it was removed from its half of the storage block, as shown on scan

line  $k+2$  in Figure 25. After all additions and deletions had been performed, the list of segments was sorted to consolidate storage. This entire process has been portrayed in Figure 26. As stated before, an edge usually separated two polygons. To eliminate redundant operations, the array  $F(2,i)$  was used to store the indices of the polygons common to edge  $i$ . When an edge entered the display and separated two polygons, its values of intersection were added to the blocks of storage for both segment lists.

This algorithm was divided into the same three parts, the Thinker, the Looker, and the Controller, as the hidden line routine. The Looker compared all segments which intersected a span and developed sufficient information for the Thinker to process it. Provided the contents of a span satisfied the categories 1 through 3 above, the Thinker was able to generate the data required for displaying this span. If the information contained in the span was too complicated, it was divided by the Controller. The successful scan line division points normally occurred at the left-most endpoint of a segment. Since this division point's location on the next scan line can be predicted, it was stored and used to decrease the time required to process the entire image. The computation of the  $X$  and  $Z$  values of a segments endpoints and an updated, sorted list of segments was also performed by the Controller. This active list of segments was sorted in the Xsort lists,  $IXSLFT(i)$  and  $IXSRGT(i)$ , by increasing  $X$  values. These two lists provided the index of the segment to the left and to the right of segment  $i$  by:

$k = IXSLFT(i)$  : was the index of the segment to the left.  
 $j = IXSRGT(i)$  : was the index of the segment to the right.

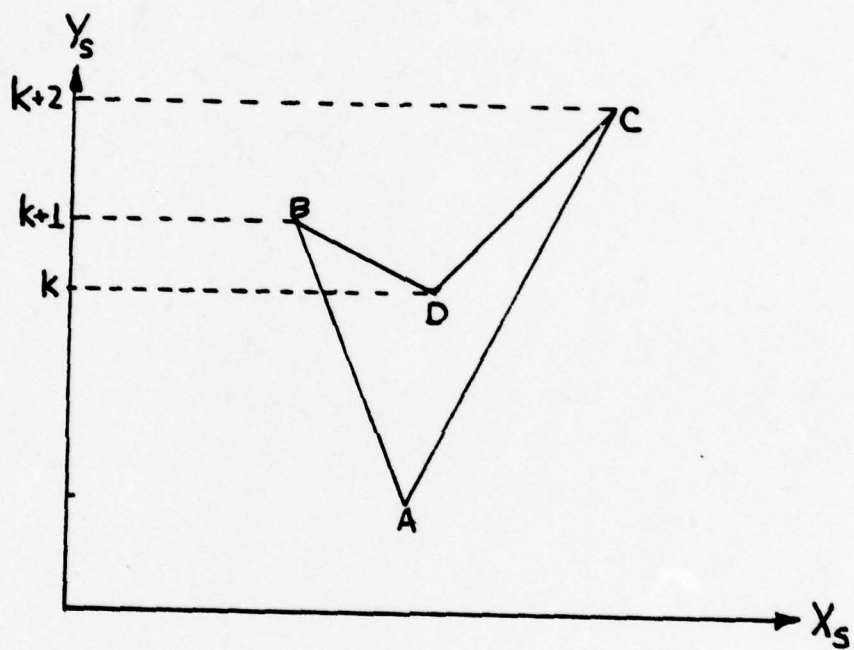


Figure 25 - ENTERING AND DEPARTING SEGMENTS

SCAN LINE	POLYGONAL SEGMENT STORAGE BLOCKS								
K	<table border="1"><tr><td>L<sub>1</sub></td><td>R<sub>1</sub></td></tr></table>	L <sub>1</sub>	R <sub>1</sub>		ONE POLYGONAL SEGMENT				
L <sub>1</sub>	R <sub>1</sub>								
K+1	1.	<table border="1"><tr><td>L<sub>1</sub></td><td>R<sub>3</sub></td></tr></table>	L <sub>1</sub>	R <sub>3</sub>	<table border="1"><tr><td>L<sub>2</sub></td><td>-</td></tr></table>	L <sub>2</sub>	-	ADD EDGE DB	
	L <sub>1</sub>	R <sub>3</sub>							
	L <sub>2</sub>	-							
2.	<table border="1"><tr><td>L<sub>1</sub></td><td>R<sub>3</sub></td></tr></table>	L <sub>1</sub>	R <sub>3</sub>	<table border="1"><tr><td>L<sub>4</sub></td><td>-</td></tr></table>	L <sub>4</sub>	-	<table border="1"><tr><td>L<sub>2</sub></td><td>-</td></tr></table> ADD EDGE DC	L <sub>2</sub>	-
L <sub>1</sub>	R <sub>3</sub>								
L <sub>4</sub>	-								
L <sub>2</sub>	-								
3.	<table border="1"><tr><td>L<sub>1</sub></td><td>R<sub>3</sub></td></tr></table>	L <sub>1</sub>	R <sub>3</sub>	<table border="1"><tr><td>L<sub>4</sub></td><td>R<sub>2</sub></td></tr></table>	L <sub>4</sub>	R <sub>2</sub>	CONSOLIDATE STORAGE		
L <sub>1</sub>	R <sub>3</sub>								
L <sub>4</sub>	R <sub>2</sub>								
K+2	1.	<table border="1"><tr><td>-</td><td>R<sub>3</sub></td></tr></table>	-	R <sub>3</sub>	<table border="1"><tr><td>L<sub>4</sub></td><td>R<sub>2</sub></td></tr></table>	L <sub>4</sub>	R <sub>2</sub>	DELETE EDGE AB	
	-	R <sub>3</sub>							
	L <sub>4</sub>	R <sub>2</sub>							
2.	<table border="1"><tr><td>-</td><td>-</td></tr></table>	-	-	<table border="1"><tr><td>L<sub>4</sub></td><td>R<sub>2</sub></td></tr></table>	L <sub>4</sub>	R <sub>2</sub>	DELETE EDGE DB		
-	-								
L <sub>4</sub>	R <sub>2</sub>								
3.	<table border="1"><tr><td>L<sub>4</sub></td><td>R<sub>2</sub></td></tr></table>	L <sub>4</sub>	R <sub>2</sub>		RETURN BLOCK TO FREE LIST				
L <sub>4</sub>	R <sub>2</sub>								

WHERE THE SUBSCRIPTS: 1 = EDGE AB

2 = EDGE AC

3 = EDGE BD

4 = EDGE CD

AND: L = LEFT AND R = RIGHT

Figure 26 - UPDATE OF SEGMENT BLOCK STORAGE

The last bookkeeping task performed by the Controller was to divide the list of segments into the following categories:

1. "SEGOUT- the right edge of the segment was contained in this span. (This segment did not appear in any span to the right of this one.)
2. SEGACT-the right edge of the segment extended beyond the right limit of this span." 8

If this span was displayed, then the SEGOUT list did not need to be considered in subsequent spans and could be discarded (until the next scan line). The SEGACT list was automatically added to the next spans active segment lists. If the span failed to be processed, the two lists were combined and compared to the new span.

When the segment lists were passed to the Looker, the X and Z coordinates of the left-most and the right-most parts of a segment in this span were computed (see Figure 27 where the following terms are portrayed:  $s_{xleft}$ ,  $s_{zleft}$ ,  $s_{xright}$ , and  $s_{zright}$ ). A box was constructed about the first segment examined in the X-Z plane which entirely surrounded that part of the segment which intersected the span. As each new segment was compared to the box, the box was enlarged to include it. If a segment completely hid the box from the viewpoint or if it was a spanner, then the count of the segments in the box was reset to one and the box was made to enclose only this segment. This was shown in Figure 28 along with the definitions of the box X and Z limits,  $b_{xleft}$ ,  $b_{zleft}$ ,  $b_{xright}$ , and  $b_{zright}$ .

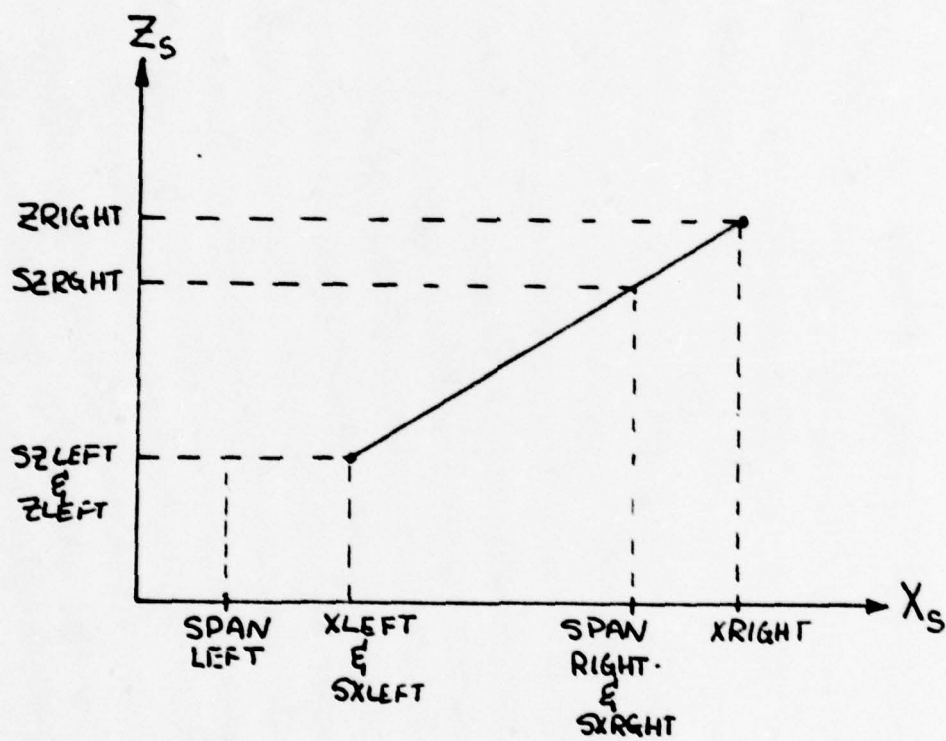
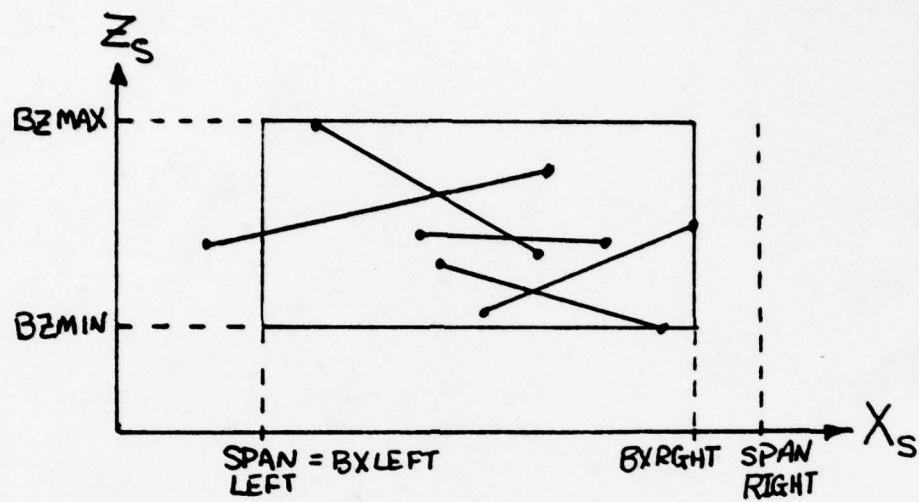
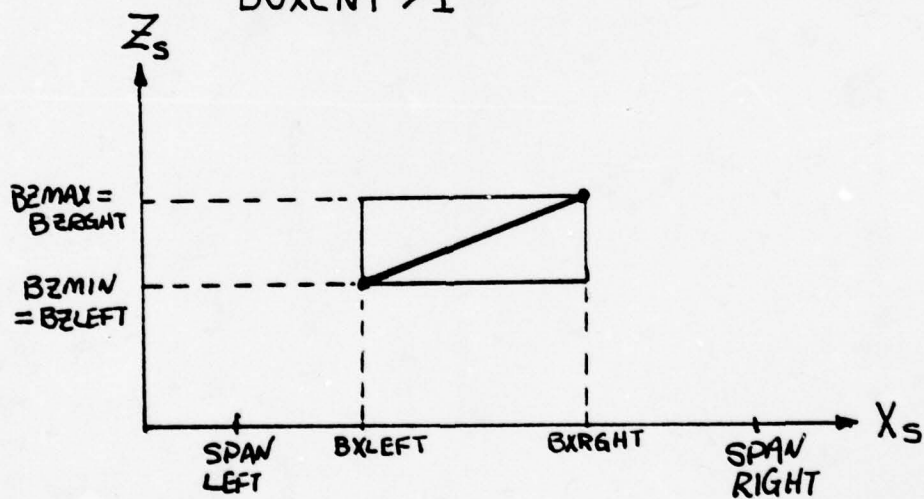


Figure 27 - SPAN DESCRIPTIVE TERMS FOR A SEGMENT



$BOXCNT > 1$

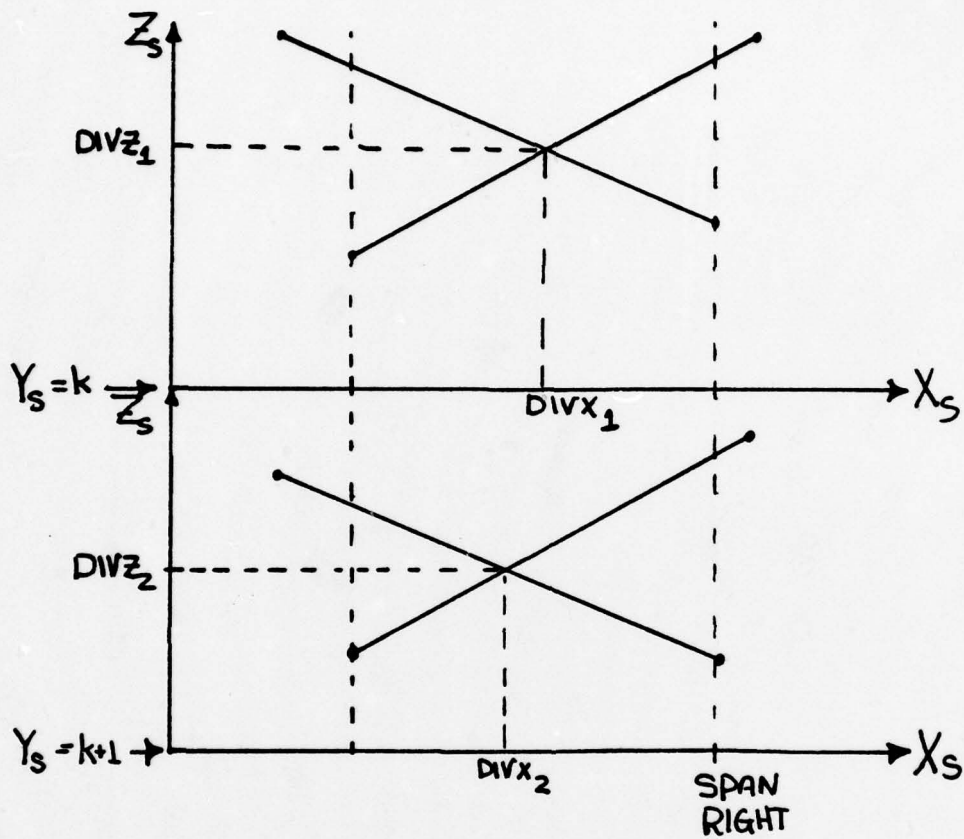


$BOXCNT = 1$

Figure 28 - SEGMENT BOX DEFINITION

The information passed to the Thinker was box count and box type. If the box count was zero, the Thinker did nothing and the Controller began processing the next span. A box count of one indicated a single segment existed in the span. The segment's *sxleft*, *sxright* and *index* were stored with the results of previous spans so that the entire scan line could be displayed at one time. When the box count was greater than one and the box type was equal to one, the span contained a simple intersection of two spanners. Both segments' *sxleft* and *sxright* values and their indices were stored with those of previous spans for this scan line. If the box count exceeded one and the box type equaled zero, the Thinker announced failure. The Controller then divided the span and began to process the left half of the old span.

An important element of all hidden surface or line elimination algorithms has been the proper display of an implied edge. When the intersection of two polygons caused an implied edge, this procedure stored the value *DIV*, the division point caused by two intersecting spanners, on its first occurrence (see Figure 29). The occurrence of this division on a second scan line provided sufficient information to compute its *X* and *Z* scan line coherence factors. The implied edge was then added to a dummy segment block and sorted with the other segments in the *Xsort* lists. However, this dummy segment block was not passed to the Looker, but was used to divide the scan line into the proper spans. When the division point due to an implied edge divided the same segment, the implied edge was discarded. A generalized flow chart was provided for this algorithm in Figure 30 to summarize this entire section.



SLOPE FOR EQUATION (1):

$$G = DIVX - DIVX_1$$

SLOPE FOR EQUATION (2):

$$C = DIVZ_2 - DIVZ_1$$

Figure 29 - IMPLIED EDGE GENERATION

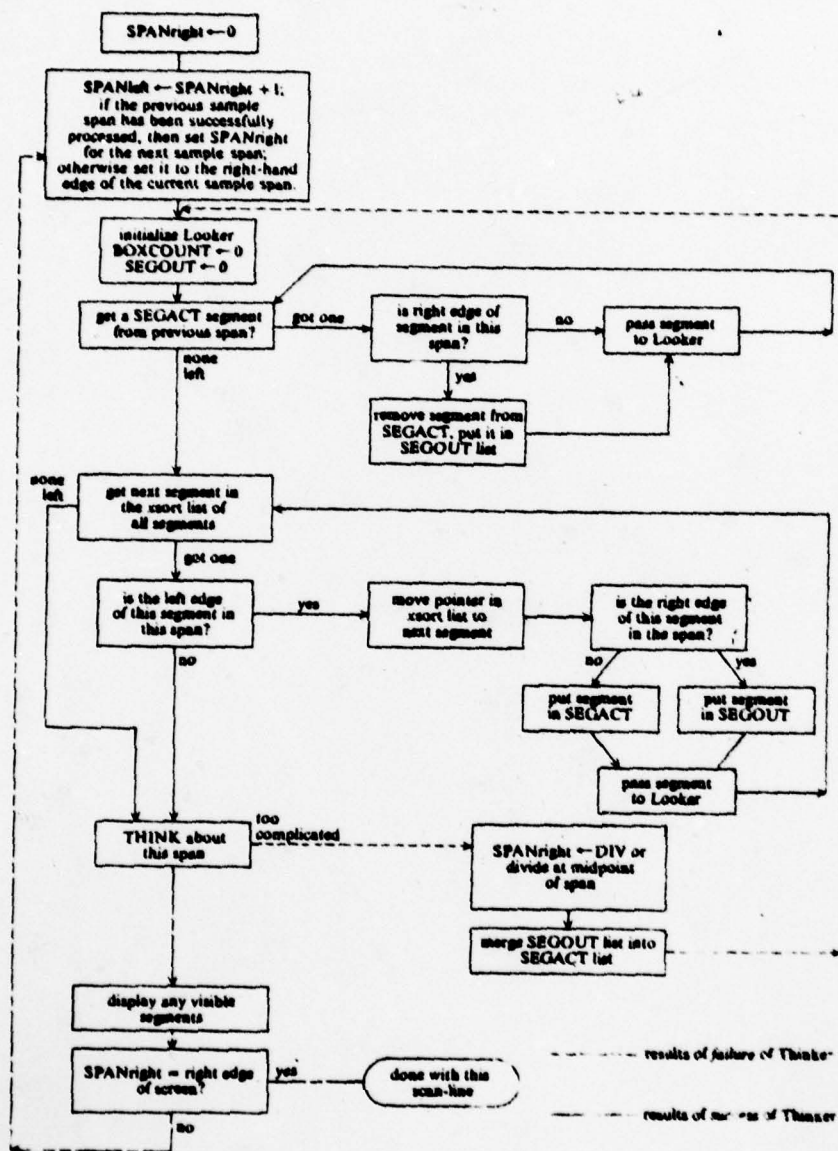


Figure 30 - HIDDEN SURFACE REMOVAL FLOW CHART

## I. IMAGE SHADING

An extremely important aspect of computer image realism was the generation of an appropriate shading algorithm. The realistic algorithms required complex software and thus, more computation time. Another alternative used has been to implement the algorithm with sophisticated and expensive hardware. Although this was one key element to a 3-D graphics language, a software implementation would have required half again as much research and time. Therefore, this aspect was left for future development. The usage of colors or shading for object definition was input via the subroutine INITIAL along with the other image data (a complete description was included in Appendix A).

#### IV. HARDWARE AND SOFTWARE CONSIDERATIONS

The realistic implementation of a standard graphics language throughout a large organization would require that the majority of the software could be input to all host computers with minimal alterations required for specific display devices. Therefore, the primary intent was to reduce display device dependence to the fewest number of subroutines. The processing of a 3-D display in real time was another important consideration. In the following sections the algorithms in III. were divided into four groups and their implementation on the display devices was presented.

##### A. IMAGE DISPLAYED - ALL LINES SHOWN

The device dependent software which had to be utilized to display an image and the different screen or display area of each machine were the two non-portable aspects of this entire graphics package. Since the intent of this effort was to produce a language which appeared to be device independent to the user, the problem of various display areas was resolved in the image data input subroutine, INITIAL. When more than one display device was supported by a host computer, the user had to select the appropriate display device number. INITIAL then chose the correct line and element resolution and the location of the display area's center. The four devices used in this project were:

1. TEKTRONIX 4012- a direct view storage tube;

2. ADAGE AGT-10- a vectored CRT;
3. VERSATEC- a hard copy device which also had electrostatic shading capability;
4. RAMTEK- a raster scan CRT.

The format of the data input by the user was stated in the graphics language description preceeding INITAL. The required data included the image description as stated in III. A. and the following information:

1. a - the distance from the screen to the viewer;
2. b - the vertical dimension of the screen;
3. the x, y, and z coordinates of the viewpoint;
4. the select display device number;
5. the index of the color table to be utilized (when using the RAMTEK);
6. and the index of the color or shading for each polygon.

A complete explanation of the input data was provided in Appendix A with the program listing of INITAL.

To display the image as input without removing hidden lines, the user then had to call the subroutine DISFLY. DISFLY called the subroutines listed below.

1. RBYCLP- performed the object to clipping coordinate transformation on the vertices and stored the results in the arrays XS(i), YS(i), and ZS(i);
2. CLIF- clipped the image against the viewable display area;
3. SCRNI- converted the clipping coordinates to screen coordinates.

The image could now be displayed by the selected device. First, each machine had to be initialized by a single device dependent subroutine call. Next, the line segments of each

edge, also called vectors, were drawn by another device dependent subroutine. Finally, all devices, except the HAMTEK, required a subroutine call which terminated the image. The ADAGE also required an image subroutine which developed a display list of vectors which was used to refresh the display screen. Thus, the device dependent portions for direct image viewing could typically be contained in three subroutines and at most four when a display list was required. The display device was initialized by the subroutine INITIAL and vectors, or line segments, were drawn by DISPLY. The graphics software package was always terminated by a call to FINISH, which when required called the device dependent subroutine to terminate the display.

The only calling parameter which was required for DISPLY was a two element integer array, IR(2), which was used to:

1. display a single polyhedron by setting  
IR(1) = IR(2) = the index of the polyhedron;
2. display a group of consecutively input polyhedra,  
where:  
IR(1) = the index of the first;  
and IR(2) = the index of the last polyhedron;
3. or display the entire image as input in INITIAL by  
setting IR(1) = 31.

Except for the TEKTRONIX, the display of a wire frame image by video graphic machines was performed in "real time". Real time was defined as a period of time so short that the user could not detect a time lag between the programs execution and the complete drawing of the image. While the computation of the image require an insignificant amount of time for the host computer of the TEKTRONIX (an IBM-360), the vector display speed of this device was very slow. The VERSATEC was also a relatively slow device, but

hard copy machines have not been expected to produce "real time" displays.

The construction of this general graphics language did omit some specific device versatilities. The VERSATEC had the capabilities to re-define its display area, resolution size, and line thickness, and could produce shaded images. These capabilities could have been added, but were left for subsequent efforts due to time constraints.

#### E. INTERACTIVE SOFTWARE AND HARDWARE

The transformations for rotation, scaling, and translation provided the means for complete image movement in three dimensions. These procedures, which have been implemented with hardware at some installations, coupled with interrupt devices, such as alphanumeric keycards, function switches, graphic tablets, joysticks, light pens, and track balls, provided a user with a complete interactive viewing capability. The subroutines written to rotate, scale, and translate a single polyhedron or the entire image were completely device independent.

The calling parameter  $IR(2)$ , which was used by each of the three subroutines, was defined and utilized as stated in A. above. Thus, new viewing aspects could be generated for a single polyhedron, a group of objects, or the entire displayable image. The usage of the scaling subroutine also required that the user provide the scale factors for the x, y, and z coordinates. The additional calling parameters required for image translation were the distances in the x, y, and z directions which the 3-D object was to be moved. The sign required for these distances was opposite to that of the standard velocity vector describing the objects

motion in this direction. Rotation calling parameters included the number specifying the axis and the angle of rotation. If an arbitrary axis was selected, the coordinates of two distinct points had to be passed into the subroutine also. The usage of these routines was defined in the comment section preceeding INITIAL. The three software transformations, rotation, translation, and scaling, required a minute amount of computation time. Any combination of a few of these three called between INITIAL and DISPLAY could be performed in "real time".

Direct view storage tube display devices have an extremely limited interactive capability due to the method used to clear the screen. This device had a writing cathode which traced the image on a fine wire mesh which was located just behind the phosphorous screen. Initially, the entire wire mesh was negatively charged. Vectors drawn on the mesh by the writing cathode caused those line segments to become positively charged. These positively charged areas accelerated and passed the electrons emitted from a second cathode, which was issuing a "flood" of electrons to refresh the image on the phosphorous screen. To clear an image, a large positive pulse was applied to the wire mesh. This caused a large flash to spread across the screen. Since the flash disrupted any possible display for several seconds, the usage of this type of device for rapidly changing, interactive images was highly unrealistic.

Vectored CRT's have achieved an extremely high degree of resolution and support most interactive interrupt devices. With this type of display machine, the image on the phosphorous screen was refreshed by storing the entire image in a vector list. A highly complex, static image, one containing several thousand vectors, could cause the display to begin flickering. This type of CRT can produce only a limited number of vectors before the phosphorous illuminated

for the first vector begins to dim. Thus, shaded images, which would require many vectors for even a simple 3-D object, can not feasibly be produced using a vectored CRT. However, excellent 3-D graphs and extremely complex wire-framed objects have been visualized, using multiple colors, with these devices.

The ADAGE AGT-10 was an extremely versatile device and had an alphanumeric keyboard, function switches, function knobs, a joystick, and a light pen as interactive capabilities. Since this machine was operated in a stand-alone mode, these interactive interrupt devices were easily utilized through the users application program, except the light pen. The light pen could only be accessed and utilized in the image subroutine. This extremely, device dependent capability was not included in the graphics software, since its usage would have required a thorough knowledge of this machines software for even a simple application.

The RAMTEK's interactive devices consisted of an alphanumeric keyboard and a set of function switches which could be used to position a cursor. The cursor's screen coordinates were obtainable through device related software. The RAMTEK's host computer was a PDP-11, which primarily supported the software "C". Since the graphics software for the RAMTEK was written in C and there was no software interface written for Fortran IV, only those subroutines required to perform hidden surface elimination were translated to Fortran. Therefore, the interactive capability of this device was limited to input via a host computer's alphanumeric terminal. The resolution of this video graphics device was 240 lines with 640 elements per line. Its lack of vertical discrimination provided poor image continuity in this dimension. Since this was the only raster scan device available at this school, its utilization

was necessary to implement the hidden surface algorithm.

### C. HIDDEN LINE REMOVAL

The hidden line removal algorithm described in Section III. was invoked by calling the subroutine REMOVE after INITIAL and any desired interactive subroutines. REMOVE called the subroutines RDYCLP, SCRIN, WARNCK, and DISPL2. The calling parameter required by REMOVE was the integer array IR(2), which was described in A. Since this algorithm clipped each edge against display windows to process an image, the vertices of the polyhedron to be displayed were passed to WARNCK expressed in un-clipped screen coordinates. WARNCK contained the Locker, the Thinker, and the Controller described in III. As the display was processed, the  $X_s$  and  $Y_s$  coordinates of the two endpoints for each vector were stored in two arrays. This storage reduced the number of device dependent subroutines added to the graphics package by this algorithm to one, DISPL2. DISPL2 generated vectors for the display exactly as performed by DISPLAY. The image subroutine used by DISPLAY to create the vector list for the ADAGE was also used by this subroutine.

The algorithm as presented in Section III. displayed each edge of a simple polyhedron, like a cube, by failure. Display by failure means that the Thinker was unable to resolve any display window and a dot was displayed when the window's size was reduced to the smallest screen resolution. An edge was displayed as a line of dots and was extremely grainy. This consistent failure had occurred because each edge was common to two polygons. When the display window was reduced so that it contained only one edge, the

intersector list still contained the indices of two polygons. The Thinker announced failure and the Controller divided the window. The computation time required to process and display a cube (which has a maximum of three viewable surfaces and nine viewable edges) exceeded twelve minutes of CPU time on the IBM-360. Additionally, almost 9,000 storage locations were required to store the endpoints of these single dot vectors.

To reduce the occurrence of display by failure, the intersector list was not rejected if only two polygons remained after comparison with the hider. When the list contained two indices, the number of edges which intersected the current display window was determined for both polygon's. Provided there was only one edge for both, the vertex indices of one polygon's edge were compared to those of the second to ensure it was the same edge. When this procedure found a common edge, the endpoints of the vector intersecting the window were stored. This addition to the Thinker reduced the CPU time to approximately 20 seconds and the storage requirements to about 300 locations.

Even though the display time had been reduced by a factor of thirty, this hidden line removal procedure was not even remotely acceptable for a real time display. While a more complex Thinker could possibly reduce the computation time, hardware implementation remains the only feasible method for a real time display using this algorithm. Additionally, the realism of wire-frame images, even with hidden lines removed, was marginal at best. As stated previously, shaded surfaces, or solid images, can not be displayed with this type of CRT. Therefore, realistic 3-D, "real time" computer graphics must be performed using a raster scan CRT with a hidden surface removal algorithm.

## D. HIDDEN SURFACE REMOVAL

The hidden surface algorithm was implemented by calling the subroutine SURFAC after INITIAL and any of the desired interactive procedures. As with all other procedures in this package, the hidden surfaces may be removed and the display created for one or a set of polyhedra or the entire image input to INITIAL as determined by the two element integer array IR(2). The object coordinate data was transformed to clipped, screen coordinates by calling the subroutines RDYCLP, CLIP, SCRIN, and SHOWIN. The remaining portion of this subroutine, SURFAC, was the Controller as described in Section III. The Looker and the Thinker were contained in the obviously named subroutines LOOKER and THINKER.

The hidden surface removal algorithm added two device dependant subroutines to the graphics software package, which were:

1. SHOWIN - constructed the desired color table and corrected the vertical coordinates;
2. SHOW - displayed each scan as its image was resolved.

The hidden line subroutines stored all of the image vectors until the entire display was resolved. However, even a single polyhedron would generate such a large list of vectors (or segments) to display solid surfaces that the storage requirements of a moderately complex scene would exceed realistic limits.

The RAMTEK, and most raster scan CRT's, could produce sixteen intensity levels for each of the three primary

colors. Thus, with all possible combinations of the shades of red, green, and blue, this device was able to display any 16 of the 4096 colors at one time. Since a shading algorithm was not implemented, SHOWIN was used to construct a color table, containing sixteen colors, which was needed for the applications program presented in Section V. Additionally, the RAMTEK's vertical scan lines (240 lines total) were twice as wide as each horizontal element of resolution. To prevent this rectangular picture element from causing image elongation, the vertical dimensions,  $V_{sy}$  and  $V_{cy}$ , were doubled. This procedure caused the horizontal,  $X_s$ , coordinates to be clipped with  $Y_s$  values which were twice their actual size. Thus, the large disparity between vertical and horizontal resolution dimensions was not allowed to cause image elongation. SHOWIN was also used to divide all  $Y_s$  coordinates by two before the controller began processing scan lines. Had these coordinate values not been divided, each scan line displayed would have been resolved and written into RAM twice.

The hardware used to input the image data onto the memory planes by the RAMTEK was called a "vector generator". Although the "vector generators" used by current raster scan CRT's display vectors at speeds only limited by the memory plane write times, the RAMTEK's generation of line segments was noticeably slow. Except for this slow vector display capability, this software algorithm developed shaded surfaces in "real time". Provided the user defined the shading or colors of each polygon, this type of display algorithm was shown to be a valuable tool for rapidly changing, realistic presentations.

AD-A081 037

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/G 9/2

A PORTABLE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE PACKAGE--ETC(U)

SEP 78 H J ROOD

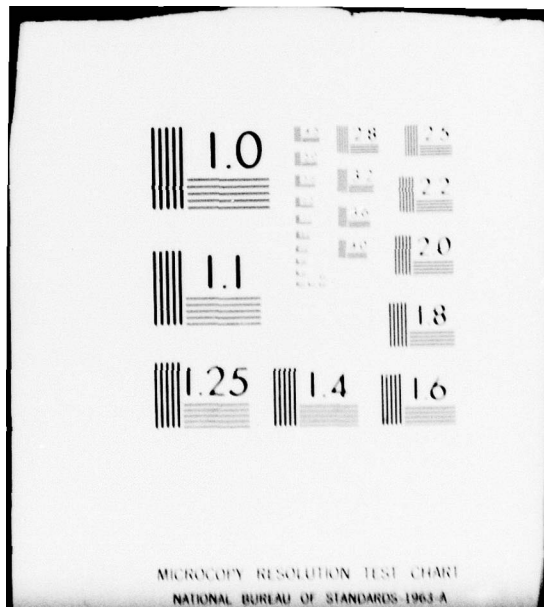
UNCLASSIFIED

NL

2 OF 3

AD  
A081037





## V. A THREE-DIMENSIONAL GRAPHICS APPLICATION

In order to demonstrate the capabilities of the 3-D Graphics Package an application program was written, which was motivated by the display of the torpedo test area at Keyport. Each test area, which was irregularly shaped, could be described by one or more convex polyhedra. A polyhedron's upper and lower surfaces represented the Puget Sound's air/water boundary and its mud bottom, respectively. Rotation, scaling, and translation of these polyhedra provided any desired viewing aspect. Since the display device at Keyport, the GENISCO GCT-3000, was a high resolution (1024 by 1024) display device, the hidden surface algorithm was selected to present a realistic 3-D display. However, the Range Safety Officer needed to see inside the polyhedral approximation of the torpedo test area not the closest polygonal surfaces. In view of this common type of display requirement, the algorithm was altered so that the hidden surfaces were displayed and the closest surfaces were deleted.

A real time display of torpedoes and other test vehicles in the range was required to prevent them from running aground or leaving the test area. Hence, an algorithm was developed which detected the polygonal surface penetration by a vehicle. Obviously, detection of an accident after occurrence would be absurd. Thus, an actual implementation of this type of display would require the construction of a safety factor by displaying surfaces which were within the actual test area boundaries. This construction would be performed by the definition of the torpedo test area when input to INITIAL and actually did not alter the display

## V. A THREE-DIMENSIONAL GRAPHICS APPLICATION

In order to demonstrate the capabilities of the 3-D Graphics Package an application program was written, which was motivated by the display of the torpedo test area at Keyport. Each test area, which was irregularly shaped, could be described by one or more convex polyhedra. A polyhedron's upper and lower surfaces represented the Puget Sound's air/water boundary and its mud bottom, respectively. Rotation, scaling, and translation of these polyhedra provided any desired viewing aspect. Since the display device at Keyport, the GENISCO GCT-3000, was a high resolution (1024 by 1024) display device, the hidden surface algorithm was selected to present a realistic 3-D display. However, the Range Safety Officer needed to see inside the polyhedral approximation of the torpedo test area not the closest polygonal surfaces. In view of this common type of display requirement, the algorithm was altered so that the hidden surfaces were displayed and the closest surfaces were deleted.

A real time display of torpedoes and other test vehicles in the range was required to prevent them from running aground or leaving the test area. Hence, an algorithm was developed which detected the polygonal surface penetration by a vehicle. Obviously, detection of an accident after occurrence would be absurd. Thus, an actual implementation of this type of display would require the construction of a safety factor by displaying surfaces which were within the actual test area boundaries. This construction would be performed by the definition of the torpedo test area when input to INITIAL and actually did not alter the display

requirements.

The location of a vehicle in the test area was provided by acoustic line of bearings which were used to determine its position. For this application, the fix provided by the bearings was assumed to generate the  $x$ ,  $y$ , and  $z$  object coordinates required for this graphics package. Since the acoustic information provided about the vehicle was not 3-D, the torpedo's location was defined as a single point. However, the actual display of the vehicle was 3-D which provided the correct relative direction of motion information (called target angle) to the observer. Although all surfaces of the torpedo were the same color, its hidden surfaces were removed for display since this algorithm required such minute computational processing time. Finally, the torpedo's track, its last five positions, was displayed as a line in the vehicles color.

#### A. PLANAR SURFACE PENETRATION

The theory used to determine if a point had penetrated one of the test area's boundaries was originally conceived by L. G. Roberts to remove hidden lines from 3-D figures. It used the coefficients of the plane equation presented in Section III. G. "These coefficients, in the vector form  $[a \ b \ c \ d]$ , were also the expression for a homogeneous vector normal to the plane (homogeneous referred to the representation of a 3-D point as a 1 by 4 vector, where  $d$  was an arbitrary scale factor). If the dot product of this normal and a vector in the viewing direction was positive, then the polygon was a back face of the polyhedron and thus, non-viewable. Face normals were also used to compute shading parameters. "

9

Additionally, if two points were on the same side of the plane of a polygon, the dot product of either point with the normal vector would have the same sign. Since concave polyhedra would allow two interior points to be on different sides of a plane determined by a polygon, only convex polyhedra were utilized. Two adjacent convex polyhedra could have at most one common polygonal face. Since penetration of this common surface by a vehicle would have falsely indicated danger, these faces were eliminated. If a vehicle penetrated a test area boundary, its color was changed to red.

#### E. HIDDEN SURFACES DISPLAYED

To present the interior of a 3-D object, it was necessary to eliminate the normally viewable surfaces. The hidden surface algorithm, which originally eliminated back planes, was easily modified to display the hidden surfaces. Only the Looker subroutine had to be altered. The Looker in Section III. compared all segments which intersected a span in order to locate one segment which was closest to the viewer and hid all other segments in the span. This subroutine was altered to search for the farthest segment from the viewer which hid all other segments from a viewpoint located on the other side of the origin (on the same viewing axis), as shown in Figure 31.

Since the majority of this image was static, the torpedo test area only required processing to display the hidden surfaces initially and when the viewing aspect was changed. However, any moving image presented a special problem on this type of display device. To project the concept of motion on a raster scan CRT, it was not sufficient to simply translate or rotate and then display the new image, because

the original figure was written in the memory planes and would be displayed until replaced. With the GENISCO, it is possible to sample the memory planes defining the color of any one pixel, but the RAMTEK, which was designed and built ten years before the GCT-3000, did not have that capability. To delete an image of a torpedo with the GENISCO, one method would be to sample the two colors displayed on the screen at the two endpoints of each line segment which defined this figure. Provided colors at these two points were the same and the vehicles image was small, this color could be stored for this segment. It could then be used to restore the original background when the torpedo's position changed. However, the time required for these background color computations could be more than that to re-process and display the entire image again. Without this option, using the RAMTEK, the static image was input to the hidden surface display algorithm and re-drawn each time the torpedo's location changed. The torpedo's 3-D form and its track were then displayed at their new screen locations.

To provide the maximum display flexibility, it was necessary to provide a means to both remove and display hidden surfaces. Thus, an additional calling parameter, ILOOK, was added to the subroutine SURFAC (the Controller). This parameter was used to enable the Controller to select the correct Looker subroutine, where:

1. ILOCK = 1, called LOOKER which removed hidden surfaces;
2. and ILOCK =2, called LOOKR<sup>1</sup> which displayed the back polygonal planes.

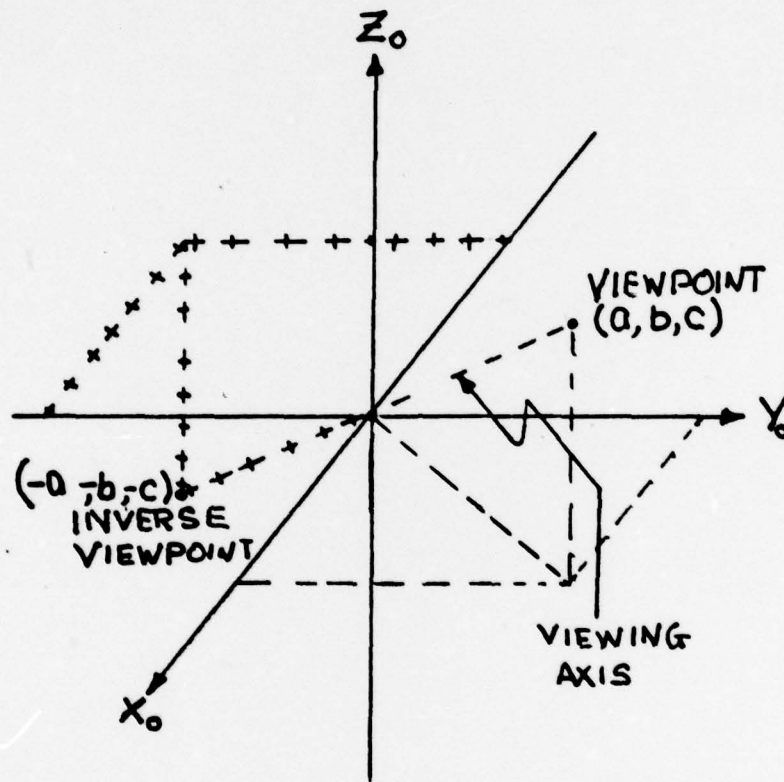


Figure 31 - VIEWPOINT REQUIRED TO DISPLAY THE BACK SURFACES

### C. TORPEDO TEST AREA SIMULATION

The torpedo test area utilized for this application program was constructed with three, box type polyhedra. The two end boxes had five polygonal surfaces and the middle had four. The test area, which was shown in Figure 32 as a wire-frame image, had all surfaces displayed as light blue, except the bottom which was light brown. The torpedoes were colored black.

Interactive input to this program provided new torpedo positions in x, y, and z object coordinates, which simulated periodic acoustic fixes and subsequent display update. Any two, distinct vehicle locations described its direction vector. When a new position was input, a new direction vector was formed between this and the last torpedo location. To compute the correct target aspect, these two direction vectors were used to find the angle through which the vehicle had rotated, as shown below:

$$\cos \theta = (\underline{a}^T \underline{b}) / (||\underline{a}|| ||\underline{b}||) ,$$

where  $\underline{a}$  and  $\underline{b}$  were the two direction vectors and the  $||\underline{a}||$  operation represented the magnitude of vector  $\underline{a}$ .

The vehicle's image was then rotated about the axis which was normal to both vectors. This normal vector was found by forming the cross product of the two direction vectors. The two points used to specify this arbitrary axis required by the subroutine ROTATE were the vehicles present position, (x y z 1), and a point found using the normal vector. These two operations were summarized algebraically as :

$\underline{N} = \underline{a} \times \underline{b}$ , where  $\times$  represents the operation of vector cross product.

and

$$(x' \ y' \ z' \ 1) = (x \ y \ z \ 1) + \underline{N}$$

Thus, the correct vehicle aspect was obtained by translation of the image to  $(x \ y \ z \ 1)$  and rotation about the arbitrary axis specified above through the angle theta.

To realistically describe a 3-D object by planar polygonal mosaics was difficult if the object was composed of curved surfaces like a torpedo. Its cigar shape was basically represented by an octagonal cylinder. The rounded nose of the torpedo was roughly approximated by reducing the diameter of the cylinder. The smaller diameter of a torpedo's tail was exaggerated by reducing the cylinder's diameter to a point. An approximation of a propeller was attached to this point. As shown by the wire-frame image in Figure 33, even a crude approximation of such a complex surface required many polygons. The actual display of this image required the definition of 27 polygons, 58 edges, and 33 vertices. While the actual numerical values were not important, they showed that the storage requirements increased rapidly with the complexity of the surfaces which were to be displayed. Simple geometric figures, such as buildings, required little storage and the effort of approximation was minimal. The realistic approximation of complex surfaces required a large amount of storage and quite a lot of artistic talent. Finally, these numerical values indicated, as observed with nearly all 3-D figures, that the memory allocations required to store edge definitions were usually double that for any other image descriptor.

The applications program, the polygonal penetration

subroutines, LOOKR1 (the modified Looker subroutine) and their respective flow charts were included in Appendix B.

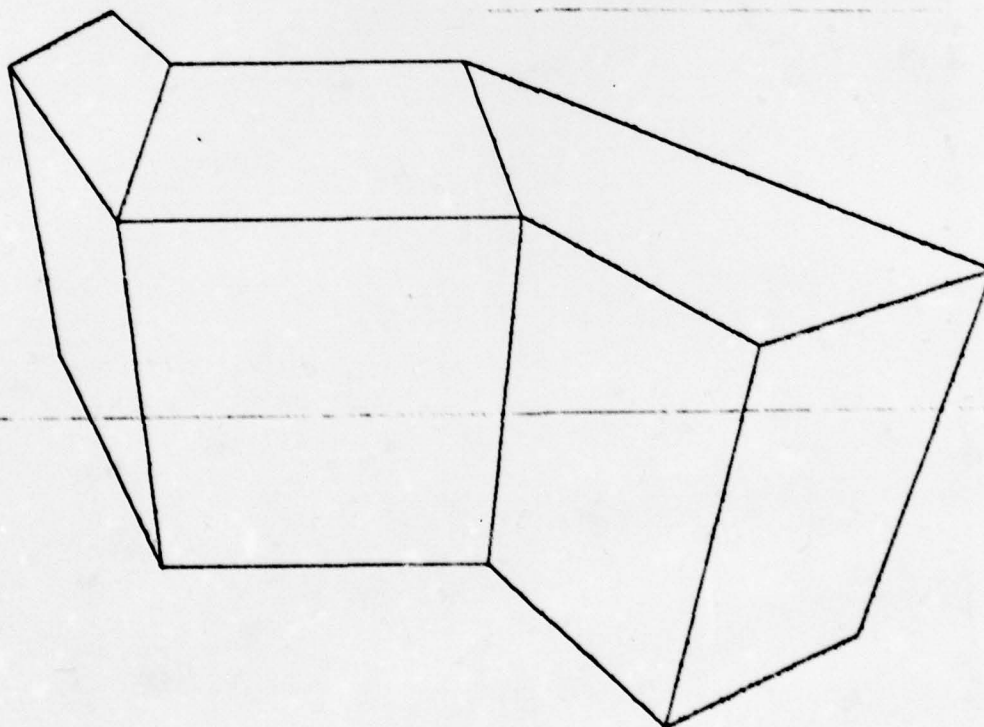
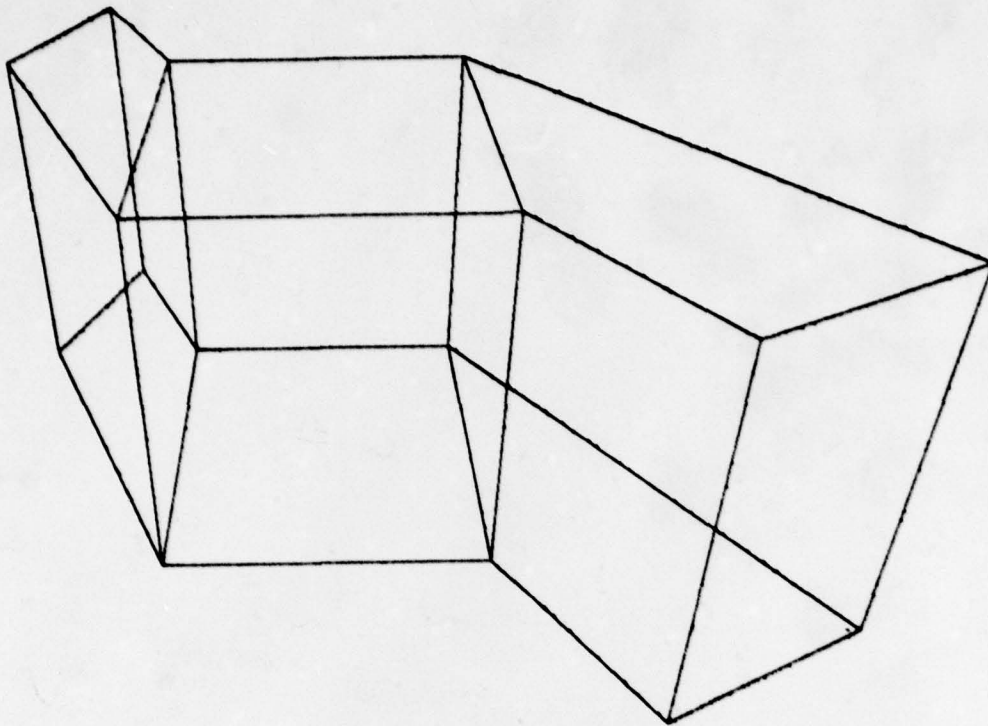


Figure 32 - TORPEDO TEST AREA

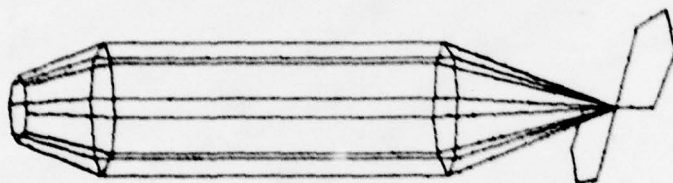


Figure 33 - TORPEDO APPROXIMATION

## I. CONCLUSIONS

The Three-Dimensional Graphics Package contained in Appendix A did not include two important aspects of computer generated graphics, an image shading procedure and 2-D and 3-D graphs. Both items are important aspects of graphical presentations and should be included in a complete package. They were items left for future research due to the length of time required to develop the present graphical software.

The intent of this effort was to provide a portable computer graphics software package. To a very real extent, this goal was accomplished. However, the development software language, Fortran IV, was not entirely portable. In fact, a program which removed hidden lines on the IBM-360 failed on the PDF-11 because of one its fortran idiosyncrasies. Additionally, an abnormal amount of time was expended attempting to input data via a file on each new computer. It would make more sense for an organization to really standardize the fortran supported by all of its main computers before a portable graphics software was implemented.

Fortran as the development language for this graphics package proved to be quite efficient with one exception. If Fortran IV had the binary operations common to languages like C and SAIL, the graphics software could have been simplified. While no time comparisons between two languages were attempted, the excellent, "real time", results achieved with the hidden surface algorithm indicated that Fortran was highly acceptable as the development language. Fortran may not be the best language available for graphics, but it is

the only universally supported and accepted software.

The 3-D graphics package, as presented in Appendices A and B, provide a user with the ability to present any object on any selected display device which is supported through Fortran. Its portability was demonstrated on four distinct types of display devices through the interface of three different host computers. There were only seven subroutines that contained statements which were device dependent. One of these, SHOWIN would not have been required if the raster scan CRT available had had square picture elements. These subroutines all call device procedures which performed the same task, but had different names. Each of the device related subroutines caused the device to generate a line segment (or vector) on the display surface. Usually, the display machines were built around a 16-Bit display processor. The primary task of the device dependent procedures was a data conversion interface between the host and the display computer. Therefore, if a large organization utilized a "standard" Fortran and required that all interface subroutines utilize standard names, this Three-Dimensional Graphics Software could be made completely portable. The obvious advantages of this software installation would be a great reduction in software and personnel training costs.

## APPENDIX A

### THREE-DIMENSIONAL GRAPHICS SUBROUTINES AND FLOW CHARTS

Each subroutine was listed with a very brief explanation of its function in the comment section preceeding the program. A specific flow chart followed each program. When the contents of a program's flow chart could not be presented on a single page, a generalized flow chart was listed first. It was followed by specific charts which amplified all blocks of the flow chart that were marked with a circled number in a lower corner, such as:

1

The subrcutines were divided into three groups, Display a Wire-Frame Image, Remove Hidden Lines, and Hidden Surfaces. The list of variables defined on pages 9 through 16 were used throughout the programs and the flow charts.

#### 1. Display a Wire-Frame Image

This group of subroutines included those to input the image data, the coordinate system transformations, the image clipping procedure, and the display subroutine for wire-frame images. Additionally, the interactive subroutines to rotate, scale, and translate an image were listed with their flow charts. The subroutines which were used to multiply matrices were included, but flow charts were not drawn due to their simplicity and lack of

applicability to the graphics software effort.

CC

C  
C THE THREE DIMENSIONAL GRAPHICS PACKAGE CONSISTS OF THE FOLLOWING  
C USER CALLABLE SUBROUTINES:

- C 1. INITIAL- INPUTS ALL DATA WHICH DEFINES THE IMAGE;  
C 2. TRANSL- TRANSLATES THE ENTIRE IMAGE OR A SINGLE POLYHEDRON;  
C 3. SCALE- SCALES THE ENTIRE IMAGE OR A SINGLE POLYHEDRON;  
C 4. ROTATE- ROTATES THE ENTIRE IMAGE OR A SINGLE POLYHEDRON;  
C 5. DISPLY- DISPLAYS THE IMAGE WITH ALL LINES DRAWN;  
C 6. REMOVE- DISPLAYS THE IMAGE AFTER REMOVING ALL HIDDEN LINES  
C 7. SURFACE- DISPLAYS THE OBJECT AS AN IMAGE WITH SOLID  
C SURFACES AFTER REMOVING THE HIDDEN OR BACK SURFACES.

C  
C THREE DIMENSIONAL IMAGES MUST BE CONSTRUCTED OF A SERIES OF  
C POLYHEDRONS (MAXIMUM OF 10). EACH POLYHEDRON MUST BE CONSTRUCTED  
C OF A SET OF CONNECTED POLYGONS (MAXIMUM OF 30 POLYGONS TOTAL).  
C EACH POLYGON IS DESCRIBED BY A SET OF 10 OR LESS CONNECTED EDGES  
C (THE MAXIMUM TOTAL NUMBER OF EDGES IS 60.) EACH EDGE IS DEFINED  
C BY TWO DISTINCT POINTS. EACH POINT (MAXIMUM OF 120) IS DEFINED  
C BY ITS X, Y, AND Z OBJECT COORDINATES. (WHERE THE OBJECT  
C COORDINATE SYSTEM IS A THREE DIMENSIONAL, RIGHT HANDED SYSTEM  
C WHICH MAY USE ANY UNIT OF MEASUREMENT.)

C  
C TO USE THIS PACKAGE THE FOLLOWING DATA MUST BE SUPPLIED BY  
C CALLING SUBROUTINE INITIAL:

- C A. USING A 414 FORMAT INPUT ,IN THIS ORDER:  
C (1) THE NUMBER OF POLYHEDRONS  
C (2) THE NUMBER OF POLYGONS  
C (3) THE NUMBER OF EDGES  
C (4) THE NUMBER OF POINTS  
C B. USING A 4G10.3 FORMAT INPUT THE X, Y, AND Z OBJECT  
C COORDINATES FOR EACH POINT. THE POINTS ARE INDEXED,  
C CONSECUTIVELY, AS THEY ARE INPUT.  
C C. USING A 214 FORMAT INPUT THE INDICES FOR THE TWO POINTS  
C WHICH DESCRIBE EACH EDGE. THE EDGES ARE INDEXED,  
C CONSECUTIVELY, AS THEY ARE INPUT.  
C D. TO DESCRIBE EACH POLYGON INPUT, IN THIS ORDER:  
C (1) USING AN I4 FORMAT, THE NUMBER OF EDGES WHICH  
C DESCRIBE THIS POLYGON.  
C (2) USING A 10I4 FORMAT, THE INDEX NUMBERS OF THE EDGES  
C WHICH DESCRIBE THIS POLYGON.  
C THE POLYGONS ARE CONSECUTIVELY INDEXED AS THEY ARE INPUT.  
C TO REDUCE STORAGE SPACE ALL OF THE POLYGONS WHICH DETERMINE  
C A POLYHEDRON MUST BE INPUT CONSECUTIVELY.  
C E. TO DESCRIBE EACH POLYHEDRON INPUT, USING A 214 FORMAT, THE  
C INDEX NUMBERS OF THE FIRST AND LAST POLYGONS  
C WHICH DESCRIBE THIS POLYHEDRON.  
C F. USING A 3G10.3 FORMAT INPUT THE OBJECT COORDINATES  
C X, Y, AND Z, OF THE POINT FROM WHICH THE IMAGE IS TO BE  
C VIEWED. THE VIEW POINT MUST BE EXTERIOR FROM ALL  
C POLYHEDRONS.  
C G. USING A 2G10.3 FORMAT INPUT THE DISTANCE FROM THE VIEWING  
C SCREEN THAT THE DISPLAY IS TO BE VIEWED AND THE VERTICAL  
C SIZE OF THE SCREEN. THESE TWO MEASUREMENTS MUST USE THE  
C SAME UNITS.  
C H. USING AN I4 FORMAT INPUT THE INDEX OF THE COLOR TABLE TO  
C BE USED TO CONSTRUCT THE IMAGE.  
C I. USING AN I4 FORMAT INPUT THE INDEX NUMBER OF THE COLOR FOR  
C EACH POLYGON. IF THE IMAGE IS TO BE DISPLAYED AS A WIRE-  
C FRAME (BY CALLING DISPLY OR REMOVE) THE ENTIRE IMAGE WILL  
C BE DRAWN USING THE COLOR OF THE FIRST POLYGON.

TO SHOW THE IMAGE WITHOUT REMOVING HIDDEN LINES CALL  
SUBROUTINE DISPLY AFTER INITIAL. DISPLY HAS NO CALLING  
PARAMETERS. TO DISPLAY THE IMAGE AFTER REMOVING ALL HIDDEN  
LINES, CALL SUBROUTINE REMOVE (NO CALLING PARAMETERS) AFTER  
INITIAL.

TO USE THE SUBROUTINES ROTATE, SCALE, AND TRANSL THE FOLLOWING  
PARAMETERS MUST BE SPECIFIED VIA THE USER'S PROGRAM:

A. ROTATE(IR, IAXIS, P1, P2, THETA), WHERE:

1. IR- IS A TWO ELEMENT ARRAY WHICH DEFINES WHICH PART  
OF THE IMAGE IS TO BE ROTATED, AS FOLLOWS:  
(A) TO ROTATE A SINGLE POLYHEDRON SET IR(1)=IR(2)=  
THE POLYHEDRON'S INDEX;  
(B) TO ROTATE SEVERAL CONSECUTIVELY INDEXED  
POLYHEDRONS SET:  
IR(1) = THE INDEX OF THE FIRST POLYHEDRON;  
IR(2) = THE INDEX OF THE LAST POLYHEDRON;  
(C) TO ROTATE THE ENTIRE IMAGE SET IR(1) = 31.  
2. IAXIS DETERMINES THE AXIS ABOUT WHICH THE POLYHEDRON  
OR ENTIRE IMAGE IS TO ROTATE, AS INDICATED:  
(A) IAXIS=0 - ROTATE ABOUT OBJECT X AXIS  
(B) IAXIS=1 - ROTATE ABOUT OBJECT Y AXIS  
(C) IAXIS=2 - ROTATE ABOUT OBJECT Z AXIS  
(D) IAXIS=3 - ROTATE ABOUT AN ARBITRARY AXIS WHICH  
MUST BE SPECIFIED USING THE ARRAYS P1 AND P2.  
3. P1(3) IS AN ARRAY WHICH CONTAINS THE X, Y, AND Z  
(OBJECT) COORDINATES OF ONE POINT ON THE ARBITRARY AXIS  
OF ROTATION, WHERE:  
(A) P1(1) IS THE X VALUE  
(B) P1(2) IS THE Y VALUE  
(C) P1(3) IS THE Z VALUE.  
4. P2(3) IS AN ARRAY CONTAINING THE X, Y, AND Z  
COORDINATES OF ANY OTHER DIFFERENT POINT ON THE  
ARBITRARY AXIS, WHERE P2(1), P2(2), AND P2(3) ARE USED  
FOR X, Y, AND Z COORDINATES, RESPECTIVELY.  
5. THETA IS THE ANGLE IN DEGREES THROUGH WHICH THE  
POLYHEDRON OR THE ENTIRE IMAGE IS TO BE ROTATED.

B. SCALE(IR, S), WHERE:

1. IR IS AN INTEGER ARRAY USED TO DETERMINE WHICH PART  
OF THE IMAGE IS TO BE SCALED AS DEFINED ABOVE FOR  
ROTATE.  
2. S(3) IS AN ARRAY WITH:  
(A) S(1) IS THE SCALE FACTOR FOR THE X COORDINATES.  
(B) S(2) IS THE SCALE FACTOR FOR THE Y COORDINATES.  
(C) S(3) IS THE SCALE FACTOR FOR THE Z COORDINATES.

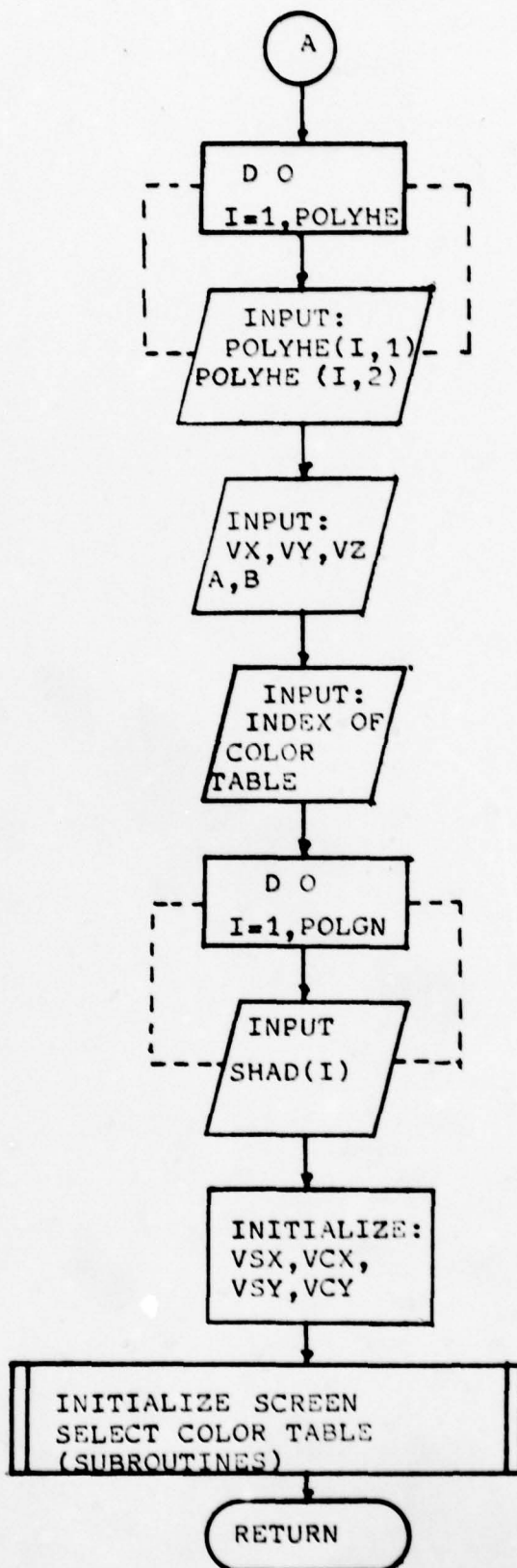
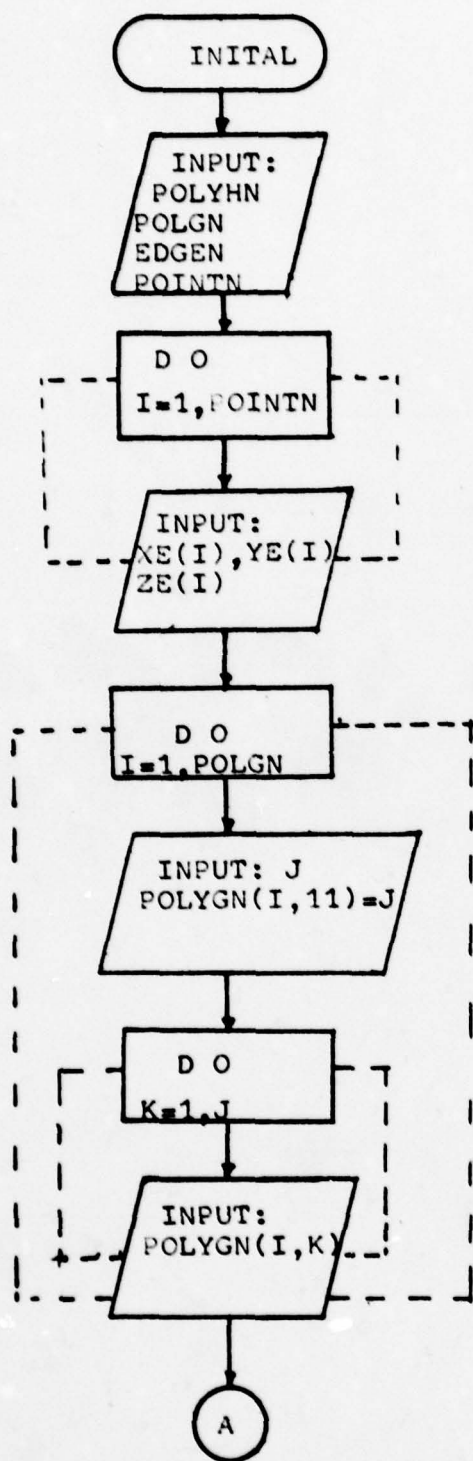
C. TRANSL(IP, T), WHERE:

1. IP IS AN INTEGER ARRAY USED TO DETERMINE WHICH PART  
OF THE IMAGE IS TO BE SCALED AS DEFINED ABOVE IN  
ROTATE.  
2. T(3) IS AN ARRAY WITH:  
(A) T(1) IS THE X DISTANCE TO TRANSLATE THE  
POLYHEDRON'S IMAGE.  
(B) T(2) IS THE Y DIST. TO TRANSLATE THE  
POLYHEDRON'S IMAGE.  
(C) T(3) IS THE Z DIST. TO TRANSLATE THE  
POLYHEDRON'S IMAGE.

THESE THREE SUBROUTINES MUST BE CALLED AFTER INITIAL AND BEFORE  
EITHER DISPLY OR REMOVE, INITIALLY. AFTER THE IMAGE IS INPUT  
THE SUBROUTINE INITIAL NEED NEVER BE CALLED AGAIN. THUS, IN



1 FORMAT(4I4)  
2 FORMAT(3G10.3)  
3 FORMAT(2I4)  
4 FORMAT(I4)  
5 FORMAT(10I4)  
6 FORMAT(2G10.3)  
  
7 FORMAT(1I)  
8 FORMAT(2X,'THE RAMTEK DEVICE WOULD NOT OPEN')  
9 FORMAT(2X,'THE FUNCTION SCREEN FAILED')  
10 FORMAT(2X,'THE FUNCTION COLORIT FAILED')  
END



```

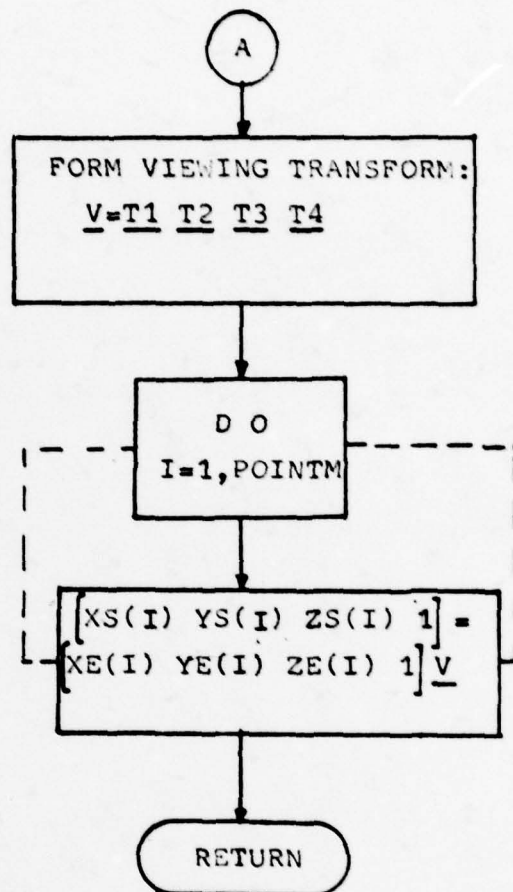
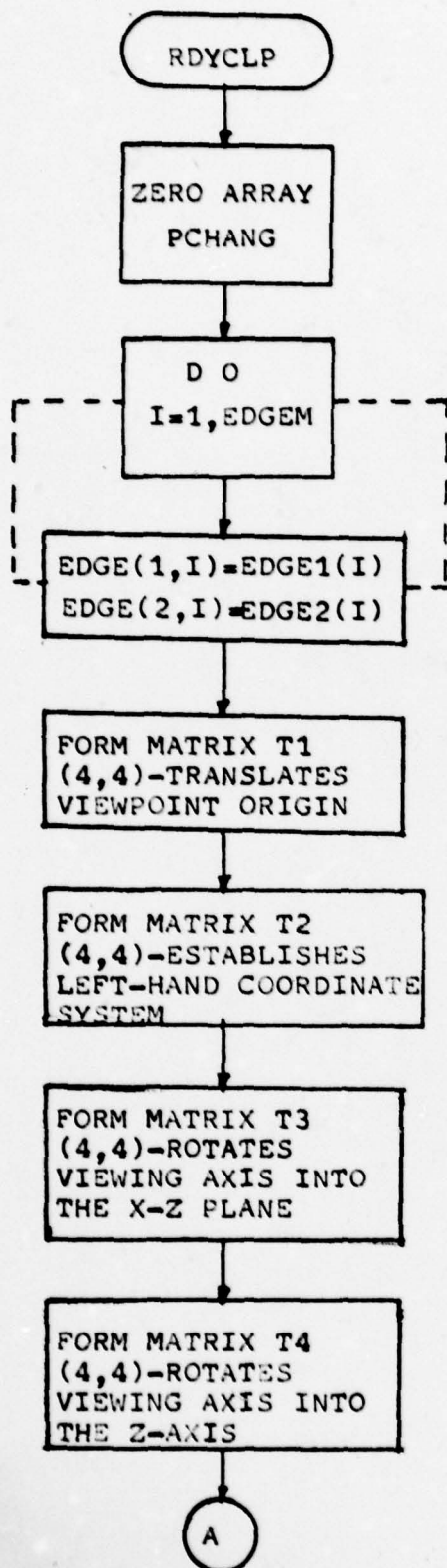
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      RDYCLP: TRANSFORMS THE OBJECT COORDINATES TO EYE COORDINATES
C      AND READYS THE IMAGE FOR CLIPPING
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE RDYCLP
  INTEGER POINTN,POINTM,EDGEN,EDGEM,EDGE(2,200),EDGE1(100)
  INTEGER PCHANG(200),EDGE2(100)
  COMMON /AAA/XE(120),YE(120),ZE(120),POINTN
  COMMON /AAH/ XS(120),YS(120),ZS(120),POINTM
  COMMON /AC/  EDGE1,EDGE2,EDGEN
  COMMON /AAD/ EDGE,EDGEM
  COMMON /CC/  VX,VY,VZ,A,B,CX,CY,CZ
  COMMON /FF/ PCHANG,THETA
  COMMON /JJ/ VSX,VSY,VZX,VZY
  DIMENSION V(4,4),T1(4,4),T2(4,4),T3(4,4),T4(4,4),RN(4,4)
  DIMENSION TNEW(4),TEMP(4)
  DATA V,T1,T2,T3,T4,RN/96*0.0/
  DO 30 I=1,120
30  PCHANG(I)=0
    POINTM=POINTN
    EDGEM=EDGEN
    DO 160 I=1,EDGEN
      EDGE(1,I)=EDGE1(I)
      EDGE(2,I)=EDGE2(I)
160  CONTINUE
    DO 136 I=1,4
      T1(I,1)=1.0
      T3(I,1)=1.0
      T4(I,1)=1.0
      RN(I,1)=1.0
136  CONTINUE
    T1(4,1)=-VX
    T1(4,2)=-VY
    T1(4,3)=-VZ
    T2(1,1)=-1.0
    T2(2,3)=-1.0
    T2(3,2)=1.0
    T2(4,4)=1.0
    SQ=VX*VX+VY*VY
    SQQ=SQRT(SQ)
    CS=VY/SQQ
    SS=VX/SQQ
    T3(1,1)=CS
    T3(1,3)=SS
    T3(3,1)=-SS
    T3(3,3)=CS
    SQ=SQ+VZ*VZ
    SRQ=SQRT(SQ)
    CSS=SQQ/SRQ
    SSS=VZ/SRQ
    T4(2,2)=CSS
    T4(2,3)=-SSS
    T4(3,2)=SSS
    T4(3,3)=CSS
    CALL GMPRD(T1,T2,V,4,4,4)
    CALL GMPRD(T3,T4,T1,4,4,4)
    CALL GMPRD(V,T1,T2,4,4,4)
    SX=1.0
    SY=1.0
    IF(VSX.GT.VSY) SX=VSY/VSX
    IF(VSX.LT.VSY) SY=VSX/VSY

```

```

RN(1,1)=A/R*SX
RN(2,2)=A/R*SY
CALL GMPRL(T2,RN,V,4,4,4)
DO 137 I=1,POINTM
    TEMP(1)=XE(I)
    TEMP(2)=YE(I)
    TEMP(3)=ZF(I)
    TEMP(4)=1.0
    CALL GPRD(TEMP,V,TNEW)
    XS(I)=TNEW(1)
    YS(I)=TNEW(2)
    ZS(I)=TNEW(3)
137 CONTINUE
RETURN
END

```



```

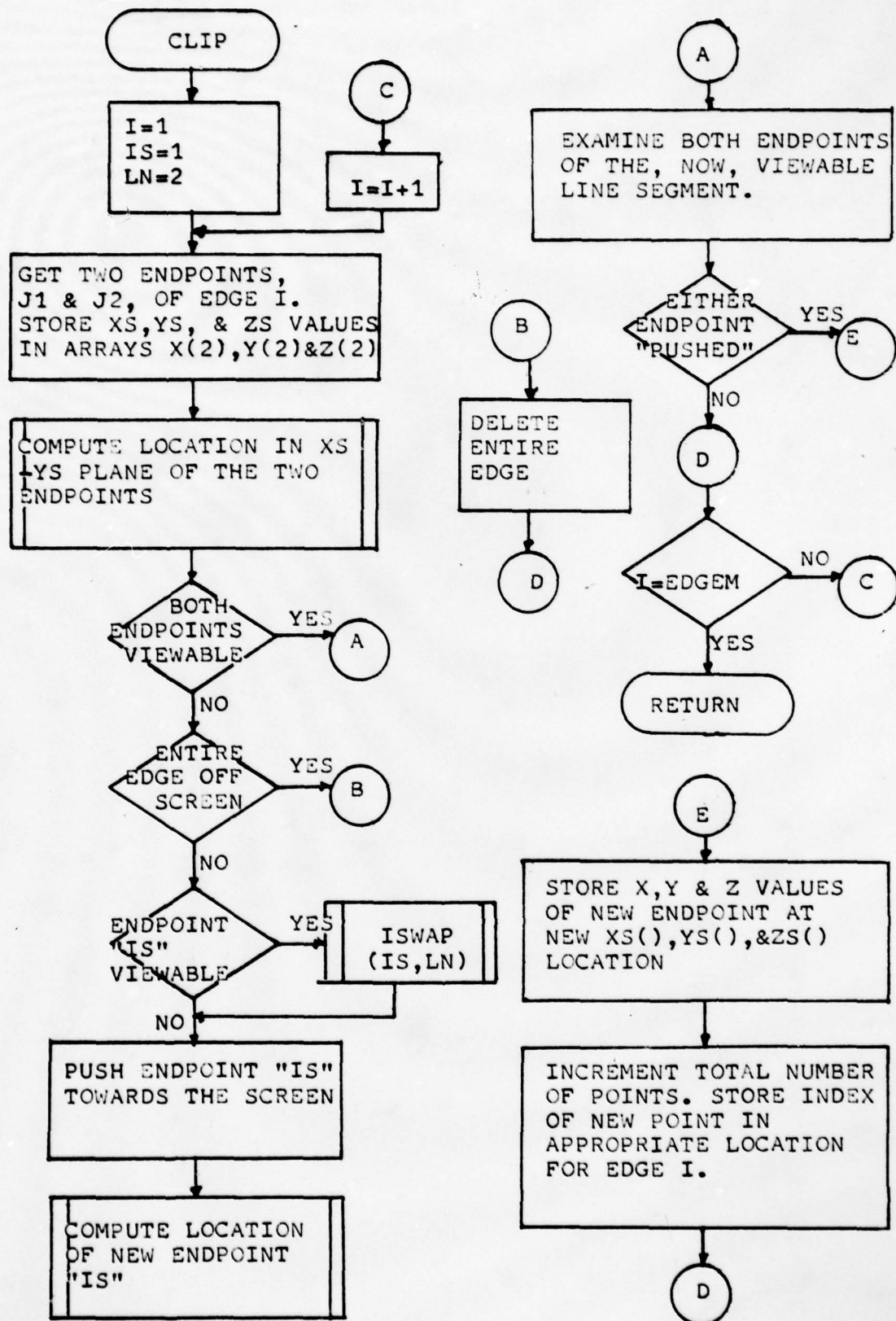
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      CLIP: CLIPS THE IMAGE AGAINST THE VISIBLE DISPLAY
C      AND ELIMINATES ALL PORTIONS OF EDGES WHICH ARE OFF THE
C      SCREEN
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE CLIP
  DIMENSION J(2),X(2),Y(2),Z(2),ITOT(2)
  COMMON /A6/ POLYGN,POLGN,SHAD
  COMMON /A4/ EDGE,EDGEN
  COMMON /F7/ PCHANG,THETA
  COMMON /AAH/ XS(120),YS(120),ZS(120),POINTN
  COMMON /I1/ ICHK(2,4)
  INTEGER POLYGN(60,11),EDGE(2,200),PCHANG(210),POLGN,EDGEN
  &,SHAD(60),POINTN
  DO 138 I=1,EDGEN
    IS=1
    LN=2
    J(1)=EDGE(1,1)
    J(2)=EDGE(2,1)
    J1=J(1)
    J2=J(2)
    X(1)=XS(J1)
    X(2)=XS(J2)
    Y(1)=YS(J1)
    Y(2)=YS(J2)
    Z(1)=ZS(J1)
    Z(2)=ZS(J2)
    ITOT(1)=ICODE(X(1),Y(1),Z(1),1)
    ITOT(2)=ICODE(X(2),Y(2),Z(2),2)
148  IR=ITOT(1)+ITOT(2)
    IF(IR.EQ.0) GO TO 140
    DO 141 K=1,4
      IA=ICBK(1,K)+ICBK(2,K)
      IF(IA.EQ.2) GO TO 142
141  CONTINUE
      IF(ITOT(1S).EQ.0) CALL ISRAP(IS,LN)
      IR=J(1S)
      PCHANG(IR)=1
      IF(ICBK(1S,1).EQ.0) GO TO 144
      I=(Z(1S)+X(1S))/(X(1S)-X(LN))-(Z(LN)-Z(1S))
      Z(1S)=I*(Z(LN)-Z(1S))+Z(1S)
      X(1S)=-Z(1S)
      Y(1S)=I*(Y(LN)-Y(1S))+Y(1S)
      GO TO 147
144  IF(ICBK(1S,2).EQ.0) GO TO 145
      I=(Z(1S)-X(1S))/(X(LN)-X(1S))-(Z(LN)-Z(1S))
      Z(1S)=I*(Z(LN)-Z(1S))+Z(1S)
      X(1S)=Z(1S)
      Y(1S)=I*(Y(LN)-Y(1S))+Y(1S)
      GO TO 147
145  IF(ICBK(1S,3).EQ.0) GO TO 146
      I=(Z(1S)+Y(1S))/(Y(1S)-Y(LN))-(Z(LN)-Z(1S))
      Z(1S)=I*(Z(LN)-Z(1S))+Z(1S)
      Y(1S)=-Z(1S)
      X(1S)=I*(X(LN)-X(1S))+X(1S)
      GO TO 147
146  I=(Z(1S)-Y(1S))/(Y(LN)-Y(1S))-(Z(LN)-Z(1S))
      Z(1S)=I*(Z(LN)-Z(1S))+Z(1S)
      Y(1S)=Z(1S)
      X(1S)=I*(X(LN)-X(1S))+X(1S)
147  ITOT(1S)=ICODE(X(1S),Y(1S),Z(1S),1S)

```

```

      GO TO 148
140  IT=J(IS)
      IF (PCHANG(IT).EQ.0) GO TO 1488
      POINTM=POINTN+1
      IF (POINTM.GT.121) GO TO 149
      XS(POINTM)=X(IS)
      YS(POINTM)=Y(IS)
      ZS(POINTM)=Z(IS)
      EDGE(IS,1)=POINTM
1488 IT=J(LN)
      IF (PCHANG(IT).EQ.0) GO TO 138
      POINTM=POINTN+1
      IF (POINTM.GT.120) GO TO 149
      XS(POINTM)=X(LN)
      YS(POINTM)=Y(LN)
      ZS(POINTM)=Z(LN)
      EDGE(LN,1)=POINTM
      GO TO 138
142  EDGE(1,1)=0
      EDGE(2,1)=0
      IT=J(1)
      PCHANG(IT)=1
      IT=J(2)
      PCHANG(IT)=1
138  CONTINUE
      RETURN
149  WRITE(6,8)
      8 FORMAT(//1X,'THE NUMBER OF POINTS HAS EXCEEDED THE MAXIMUM
      STORAGE ALLOCATED FOR POINTS'//)
      RETURN
      END

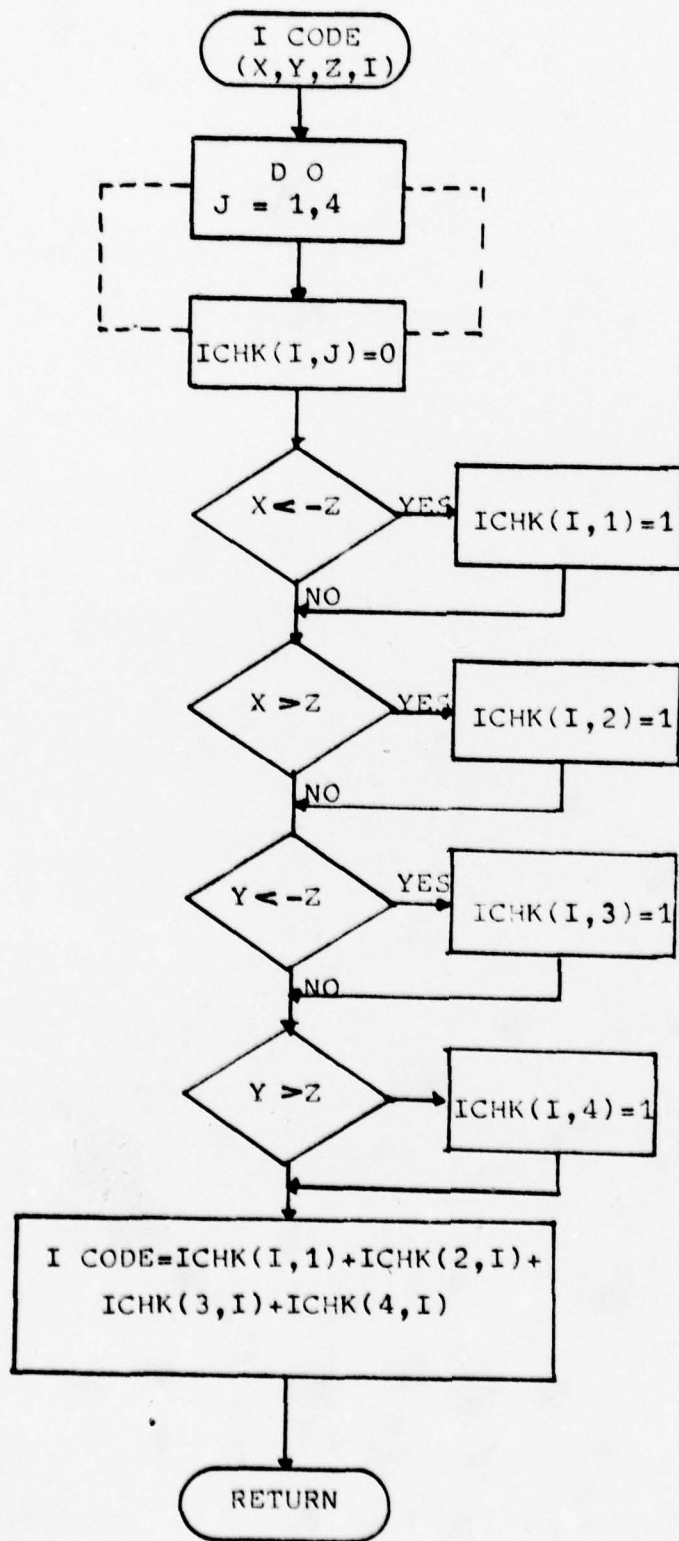
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      ICODE: USED BY CLIP TO DETERMINE IF AN END POINT IS VISIBLE AND
C      IF NOT TO WHICH SIDE OF THE DISPLAY THE POINT IS LOCATED.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION ICODE(X,Y,Z,I)
      COMMON /I1/ICLK(2,4)
      DO 139 J=1,4
139  ICLK(I,J)=0
      IF(X.LT.-Z) ICLK(I,1)=1
      IF(X.GT.Z) ICLK(I,2)=1
      IF(Y.LT.-Z) ICLK(I,3)=1
      IF(Y.GT.Z) ICLK(I,4)=1
      ICODE=ICLK(1,1)+ICLK(I,2)+ICLK(I,3)+ICLK(I,4)
      RETURN
      END

```





ISWAP(I,J)

IT=J  
J =I  
I =IT

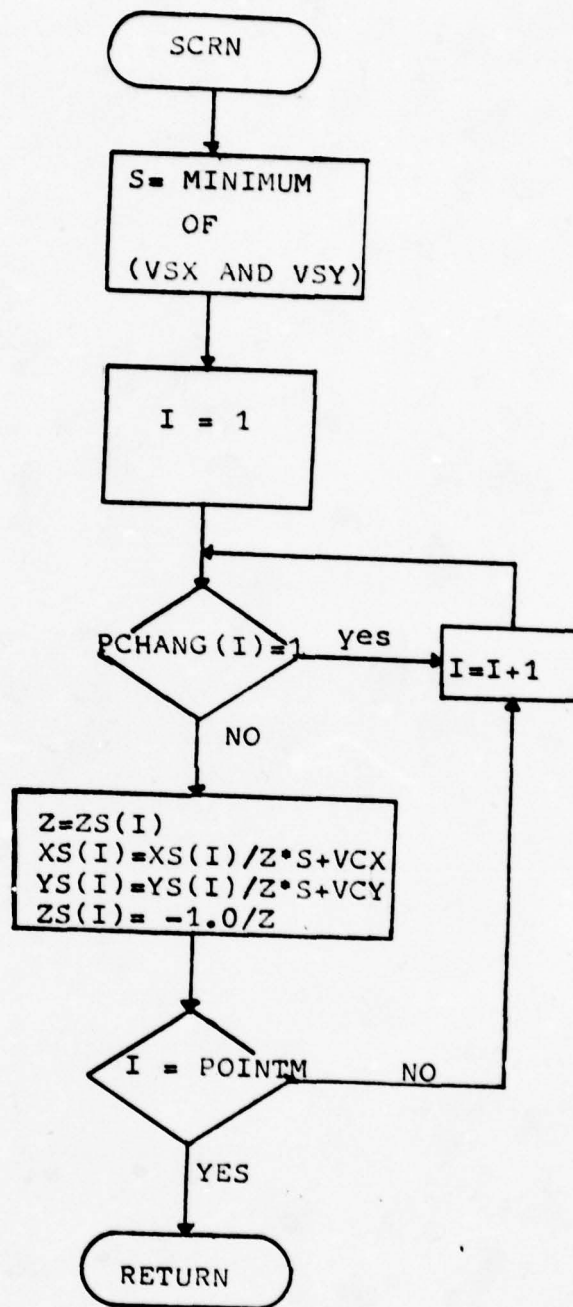
RETURN

SWAP(X,Y)

T=X  
X=Y  
Y=X

RETURN

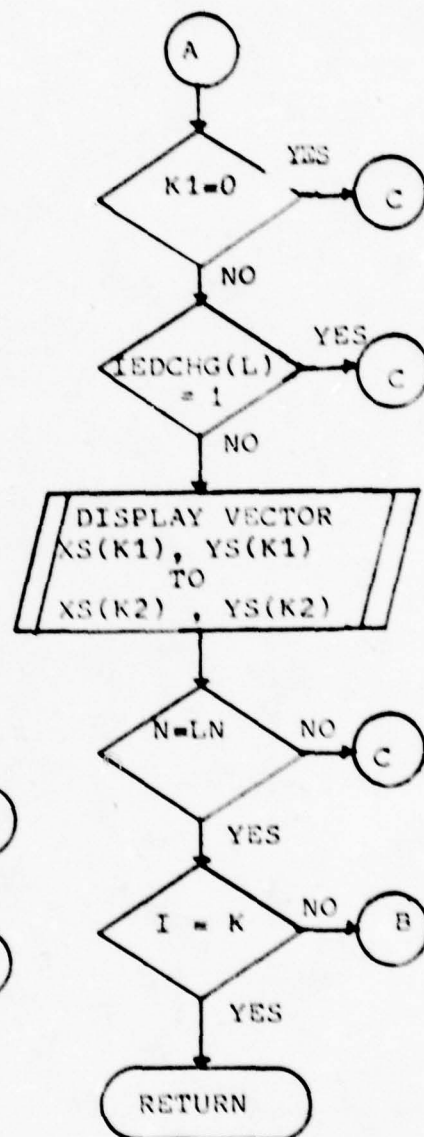
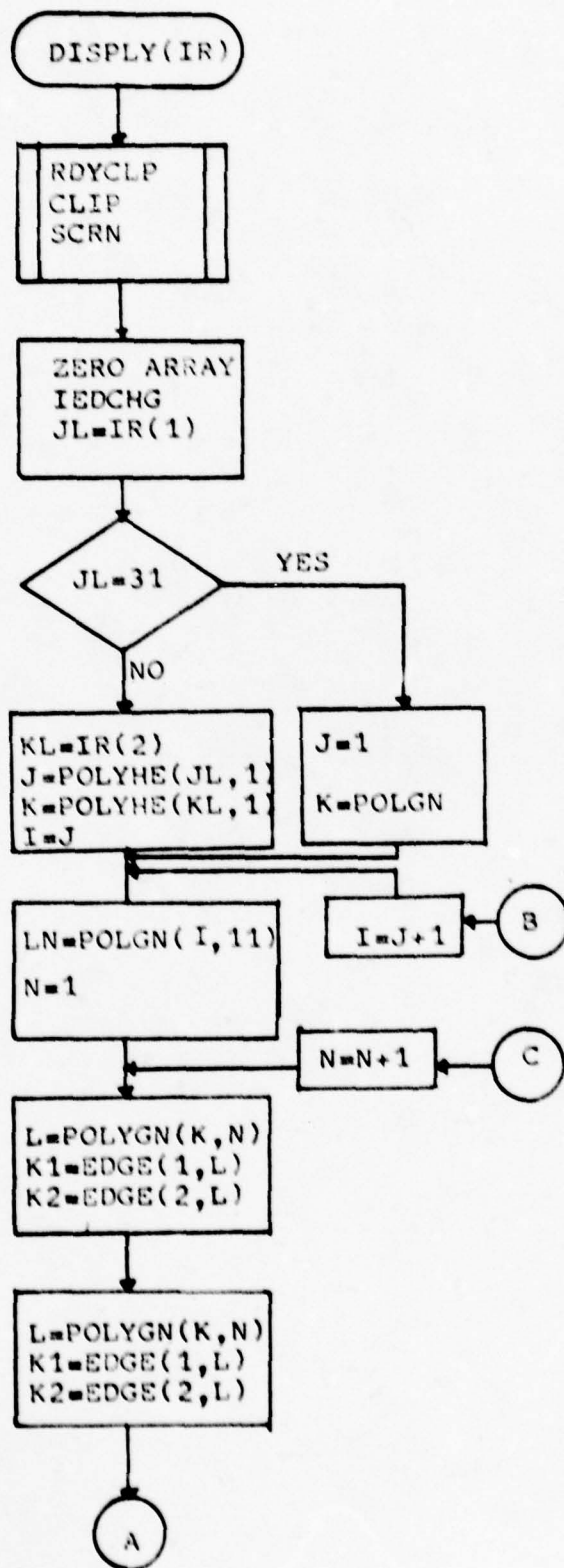




```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   DISPLY: MASTER SUBROUTINE WHICH DRAWS THE CLIPPED IMAGE ON THE
C   SELECTED OUTPUT DEVICE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE DISPLY(IR)
  DIMENSION IR(2)
  COMMON /AA/POLYHE,POLYHN
  COMMON /AB/POLYGN,POLGN,SHAD
  COMMON /AAD/ EDGE,EDGEN
  COMMON /AAH/ XS(120),YS(120),ZS(120),POINTM
  COMMON /FF/ EDGCHG(200)
  INTEGER POLYGN(60,11),EDGE(2,200),POLGN,EDGEN,COLOR
  INTEGER VECTOR,SHAD(60),POINTM,EDGCHG,POLYHE(10,2),POLYHN
  CALL RDCPLP
  CALL CLIP
  CALL SCRN
  DO 30 I=1,EDGEN
30  EDGCHG(I)=0
    JL=IR(1)
    IF(JL.EQ.31) GO TO 152
    KL=IR(2)
    I=POLYHE(JL,1)
    J=POLYHE(KL,2)
    GO TO 1522
152  I=1
    J=POLGN
1522 JLM=SHAD(I)
    II=COLOR(JLM)
    DO 1530 K=I,J
      LN=POLYGN(K,11)
      DO 1540 N=1,LN
        L=POLYGN(K,N)
        K1=EDGE(1,L)
        K2=EDGE(2,L)
        IF(K1.EQ.0) GO TO 1540
        IF(EDGCHG(L).EQ.1) GO TO 1540
        EDGCHG(L)=1
        XS1=XS(K1)
        YS1=YS(K1)/2.0
        XS2=XS(K2)
        YS2=YS(K2)/2.0
        KR=VECTOR(XS1,YS1,XS2,YS2)
        IF(KR.LT.0) WRITE(6,3)
1540  CONTINUE
1530  CONTINUE
      RETURN
3     FORMAT('THE FUNCTION VECTOR FAILED')
      END

```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   ROTATE: ROTATES A SINGLE POLYHEDRON OR THE ENTIRE IMAGE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE ROTATE(IN,RAXIS,P1,P2,THE)
COMMON /AA/ POLYHE, POLYHN
COMMON /AB/ POLYGN,POLGN,SHAD
COMMON /AC/ EDGE1,EDGE2,EDGEN
COMMON /AAA/XE(120),YE(120),ZE(120),POINTN
COMMON /FF/PCHANG,THEIA
DIMENSION I(4,4),R1(4,4),R2(4,4),R3(4,4),I1(4,4),TEMP(4,4),
&LM(4),LN(4),P1(3),P2(3),IR(2)
INTEGER POLYHE(10,2),POLYGN(60,11),EDGE1(100),EDGE2(100),
&POLYHN,POLGN,EDGEN,POINTN,RG111,RAXIS,SHAD(60),PCHANG(200)
DATA I,R1,R2,R3,I1,TEMP/96*0.0/
DO 30 I1=1,POINTN
30 PCHANG(I1)=0
THEIA=THE*3.14159/180.
JL=IR(1)
IF(JL.EQ.31)GO TO 111
KL=IR(2)
J=POLYHE(KL,2)
I1=POLYHE(JL,1)
GO TO (112,113,114),RAXIS
DO 115 K=11,J
LN=POLYGN(K,I1)
DO 116 N=1,LN
L=POLYGN(K,N)
CALL ROTX(EDGE1(L))
CALL ROTX(EDGE2(L))
116 CONTINUE
115 CONTINUE
RETURN
112 DO 118 K=11,J
LN=POLYGN(K,I1)
DO 119 N=1,LN
L=POLYGN(K,N)
CALL ROTY(EDGE1(L))
CALL ROTY(EDGE2(L))
119 CONTINUE
118 CONTINUE
RETURN
113 DO 120 K=11,J
LN=POLYGN(K,I1)
DO 121 N=1,LN
L=POLYGN(K,N)
CALL ROTZ(EDGE1(L))
CALL ROTZ(EDGE2(L))
121 CONTINUE
120 CONTINUE
RETURN
114 DX=P2(1)-P1(1)
DY=P2(2)-P1(2)
DZ=P2(3)-P1(3)
DIEMP=DX*DX+DY*DY+DZ*DZ
DIST=SQRT(DIEMP)
EX=DX/DIST
EY=DY/DIST
EZ=DZ/DIST
V=SQRT(EY*EY+EZ*EZ)
DO 122 L=1,4
I(L,L)=1.0

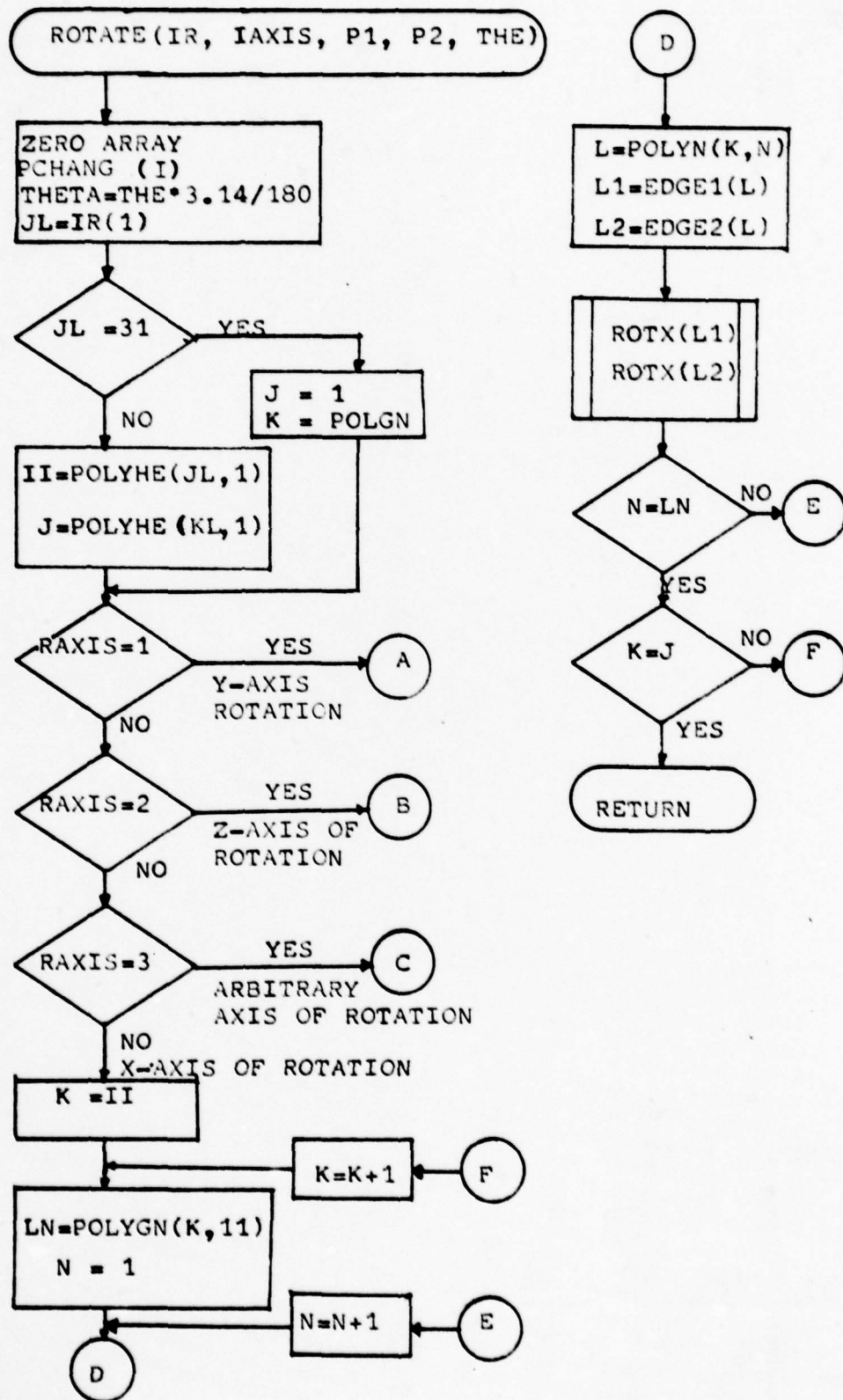
```

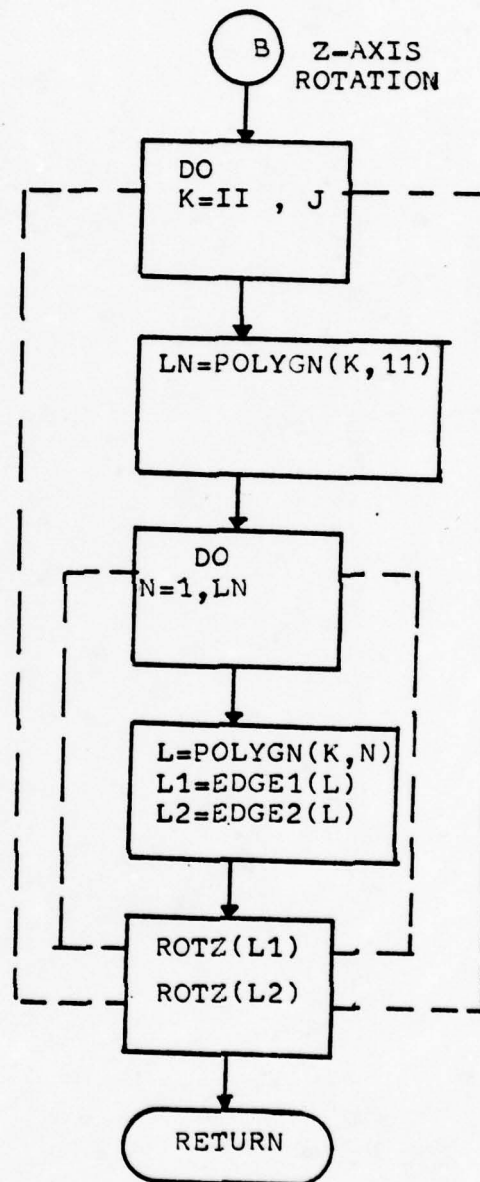
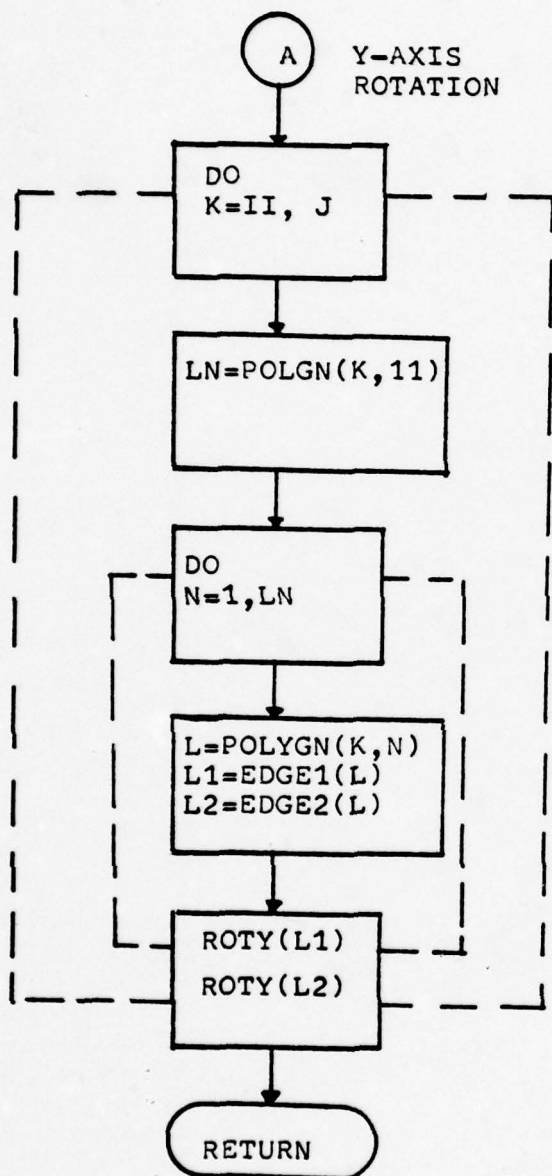
```

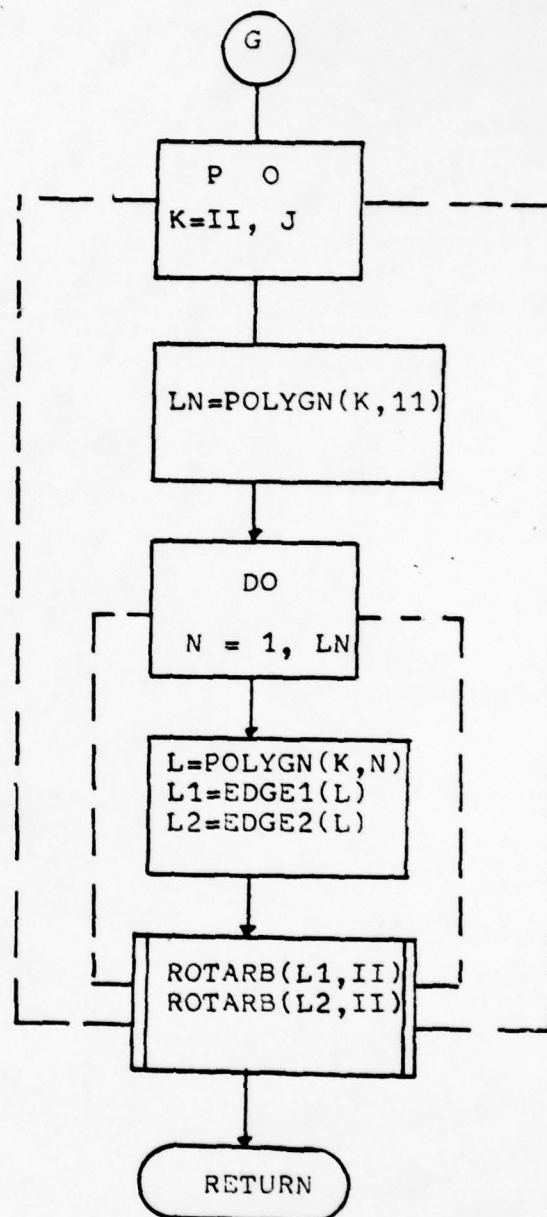
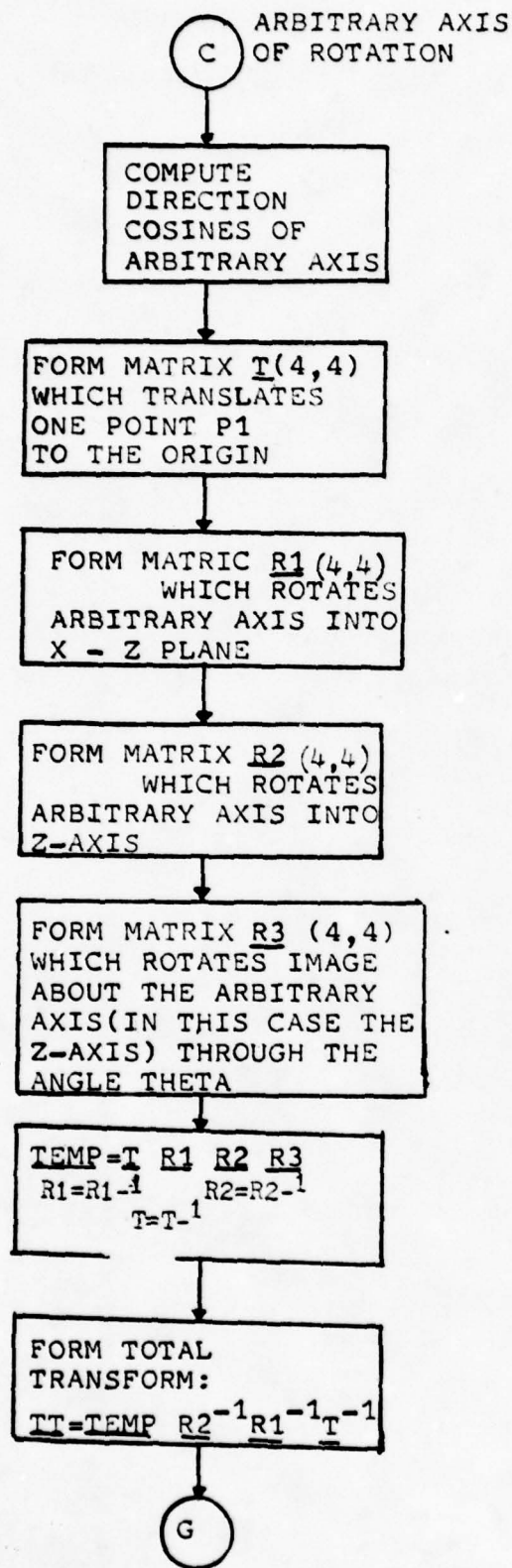
      R1(L,L)=1.0
      R2(L,L)=1.0
      R3(L,L)=1.0
122  CONTINUE
      AA=E7/V
      HH=EY/V
      T(4,1)=-P1(1)
      T(4,2)=-P1(2)
      T(4,3)=-P1(3)
      R1(2,2)=AA
      R1(2,3)=BB
      R1(3,2)=-BB
      R1(3,3)=AA
      R2(1,1)=V
      R2(1,3)=EX
      R2(3,1)=-EX
      R2(3,3)=V
      R3(1,1)=COS(THETA)
      R3(1,2)=-SIN(THETA)
      R3(2,1)=-R3(1,2)
      R3(2,2)=R3(1,1)
      CALL GMPRO(T,R1,TEMP,4,4,4)
      CALL GMPRO(TEMP,R2,11,4,4,4)
      CALL GMPRO(11,R3,TEMP,4,4,4)
      RE=V+V+EX*EX
      RF=V/RE
      RG=EX/RE
      RD=AA*AA+BB*BB
      RA=AA/RD
      RB=BB/RD
      T(4,1)=P1(1)
      T(4,2)=P1(2)
      T(4,3)=P1(3)
      R1(2,2)=RA
      R1(2,3)=-RB
      R1(3,2)=RB
      R1(3,3)=RA
      R2(1,1)=RF
      R2(1,3)=-RG
      R2(3,1)=RG
      R2(3,3)=RF
      CALL GMPRO(TEMP,R2,11,4,4,4)
      CALL GMPRO(11,R1,TEMP,4,4,4)
      CALL GMPRO(TEMP,1,11,4,4,4)
      IF(1.EQ.31) GO TO 123
      DO 124 K=11,J
        LN=POLYGN(K,11)
        DO 125 N=1,LN
          L=POLYGN(K,N)
          CALL ROTARR(EDGE1(L),11)
          CALL ROTARR(EDGE2(L),11)
125  CONTINUE
124  CONTINUE
      RETURN
123  DO 126 K=1,POINTN
      CALL ROTARR(K,11)
126  CONTINUE
      RETURN
111  GO TO (128,129,114),RAXIS
      DO 127 K=1,POINTN
        CALL ROTX(K)
127  CONTINUE
      RETURN

```

128 DO 130 K=1,POINTN  
CALL RUTY(K)  
130 CONTINUE  
RETURN  
129 DO 131 K=1,POINTN  
CALL RUTZ(K)  
131 CONTINUE  
RETURN  
END



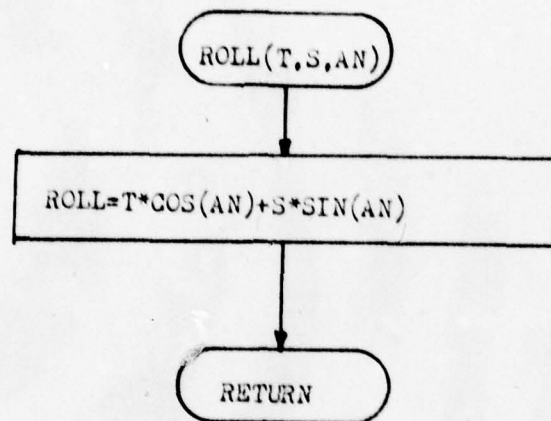




```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   ROLL: USED BY ROTX, RUTY, RUTZ, AND ROTARR TO PERFORM A REPEATED
C   CALCULATION, THE COSINE AND SINE OF THE ANGLE OF ROTATION.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
FUNCTION ROLL(T,S,AN)
  ROLL=1+COS(AN)+S*SIN(AN)
  RETURN
END

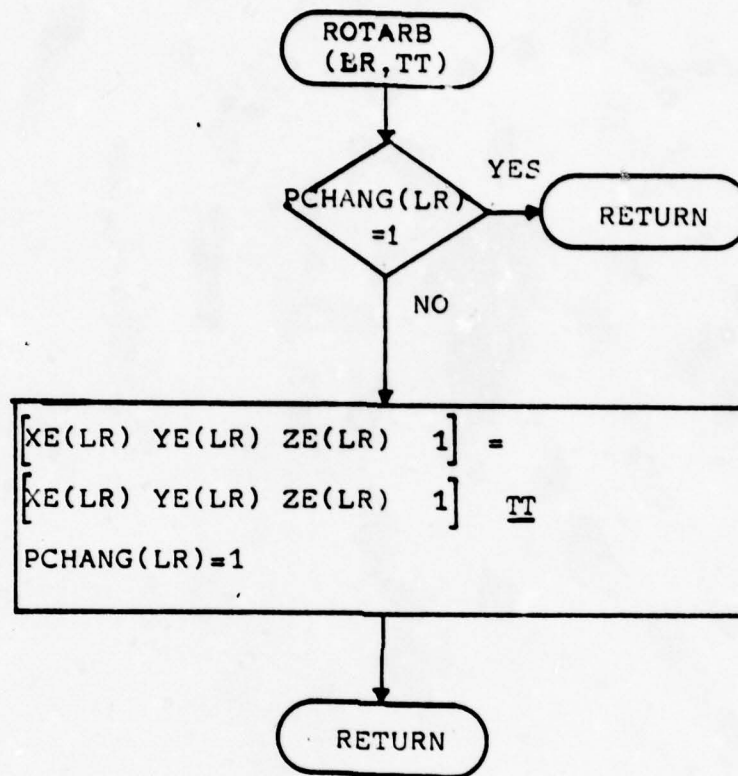
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   ROTARB: PERFORMS THE REPEATED MULT.'S TO ROTATE THE SELECTED
C   IMAGE ABOUT AN ARBITRARY AXIS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE ROTARB(LR,IT)
  DIMENSION IT(4,4),TEMP(4),TNEW(4)
  COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
  COMMON /FF/ PCHANG,THETA
  INTEGER PCHANG(200),POINTN
  IF(PCHANG(LR).EQ.1) RETURN
  TEMP(1)=XE(LR)
  TEMP(2)=YE(LR)
  TEMP(3)=ZE(LR)
  TEMP(4)=1.0
  CALL GPRD(TEMP,IT,TNEW)
  XE(LR)=TNEW(1)
  YE(LR)=TNEW(2)
  ZE(LR)=TNEW(3)
  PCHANG(LR)=1
  RETURN
END

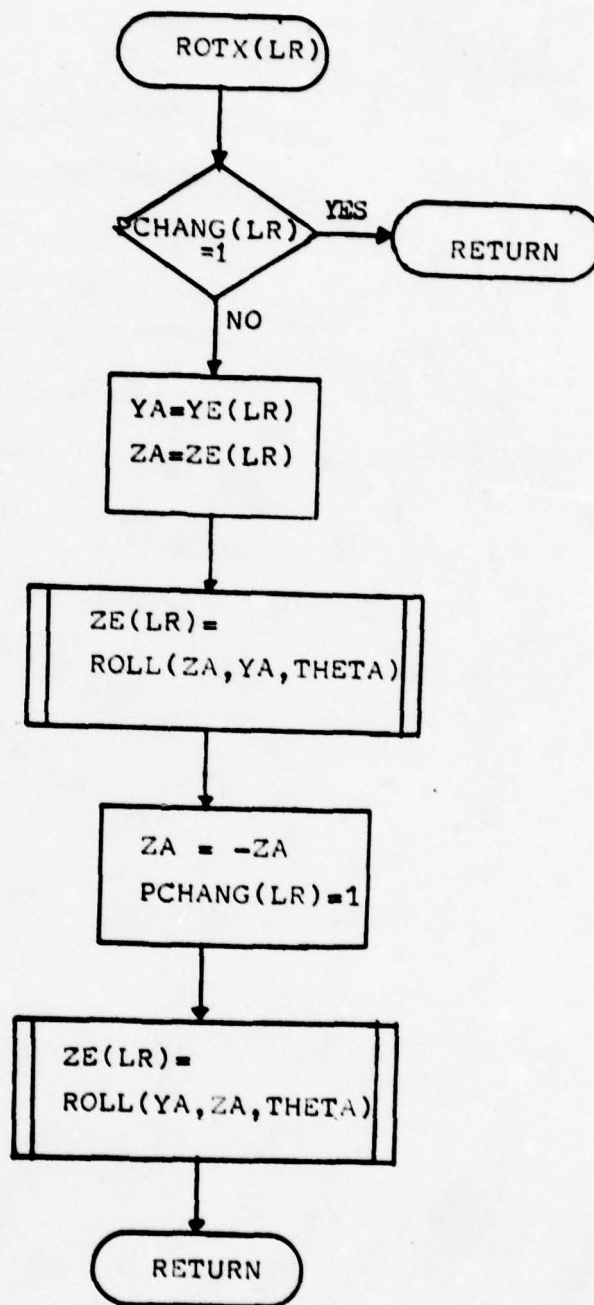
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      ROTX: PERFORMS THE REPEATED MULTIPLICATIONS TO ROTATE THE
C      SELECTED IMAGE ABOUT THE X AXIS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE ROTX(LR)
      COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
      COMMON /FF/ PCHANG,THETA
      INTEGER PCHANG(200),POINTN
      IF(PCHANG(LR).EQ.1) RETURN
      YA=YE(LR)
      ZA=ZE(LR)
      ZE(LR)=ROLL(ZA,YA,THETA)
      ZA=-ZA
      YE(LR)=ROLL(YA,ZA,THETA)
      PCHANG(LR)=1
      RETURN
      END

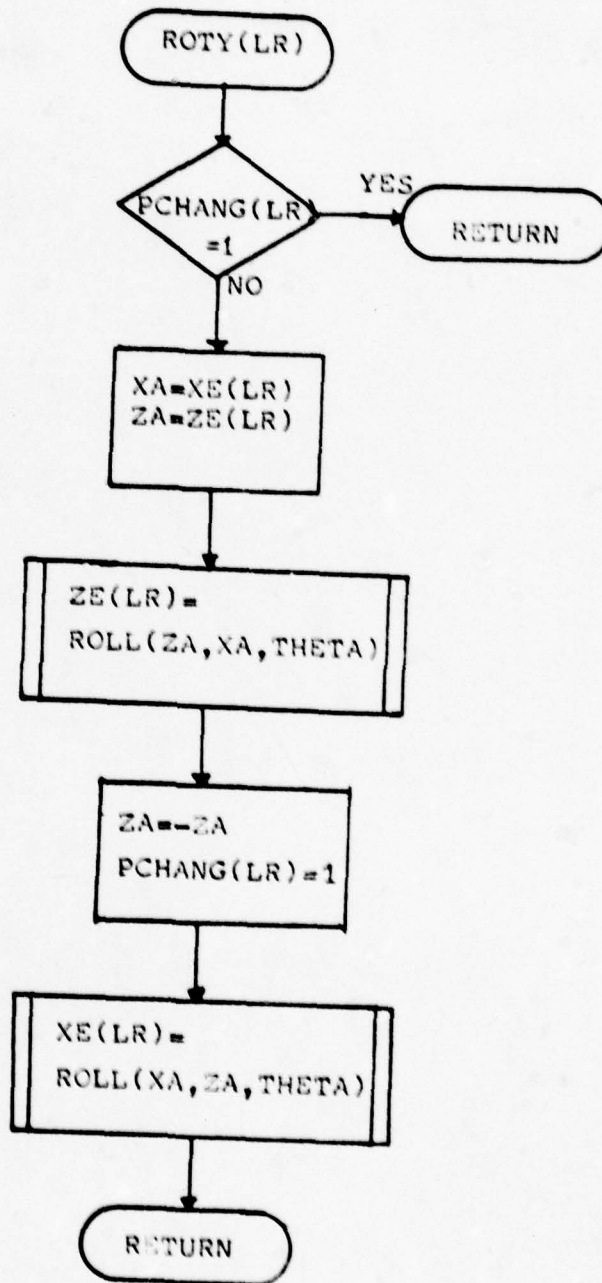
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      ROTY: PERFORMS THE REPEATED MULT.'S TO ROTATE THE SELECTED
C      IMAGE ABOUT THE Y AXIS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE ROTY(LR)
      COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
      COMMON /FF/ PCHANG,THETA
      INTEGER PCHANG(200),POINTN
      IF(PCHANG(LR).EQ.1) RETURN
      XA=XE(LR)
      ZA=ZE(LR)
      ZE(LR)=ROLL(ZA,XA,THETA)
      ZA=-ZA
      XE(LR)=ROLL(XA,ZA,THETA)
      PCHANG(LR)=1
      RETURN
      END

```



CC

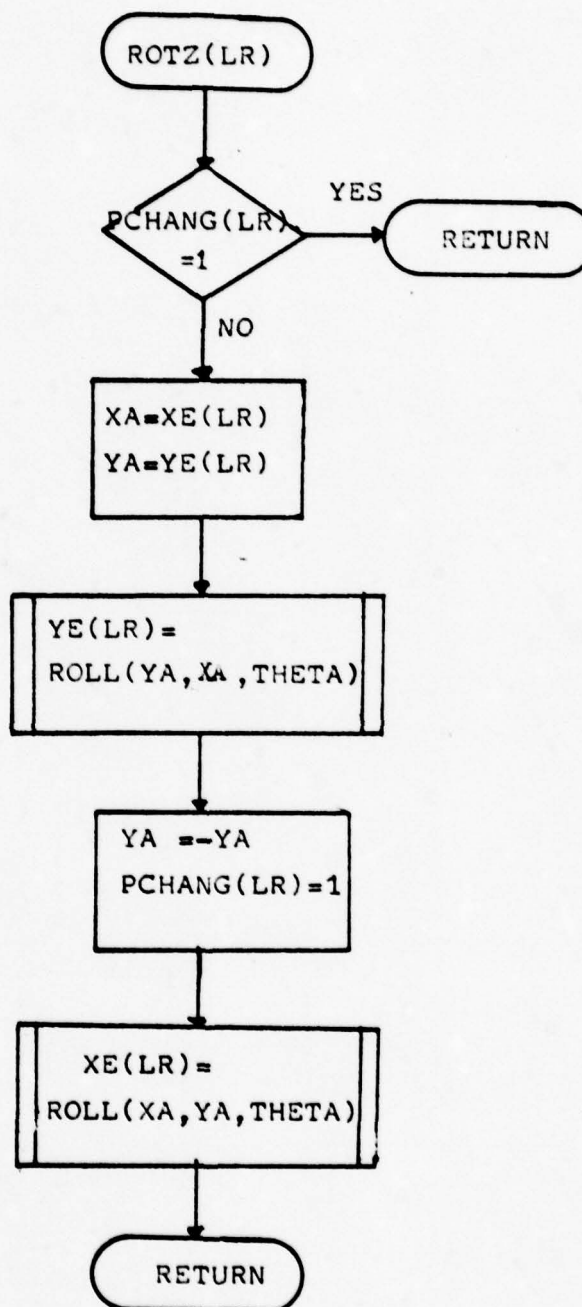
C

C     ROTZ: PERFORMS THE REPEATED MULTI.'S TO ROTATE THE SELECTED  
C     IMAGE ABOUT THE Z AXIS

C

CC

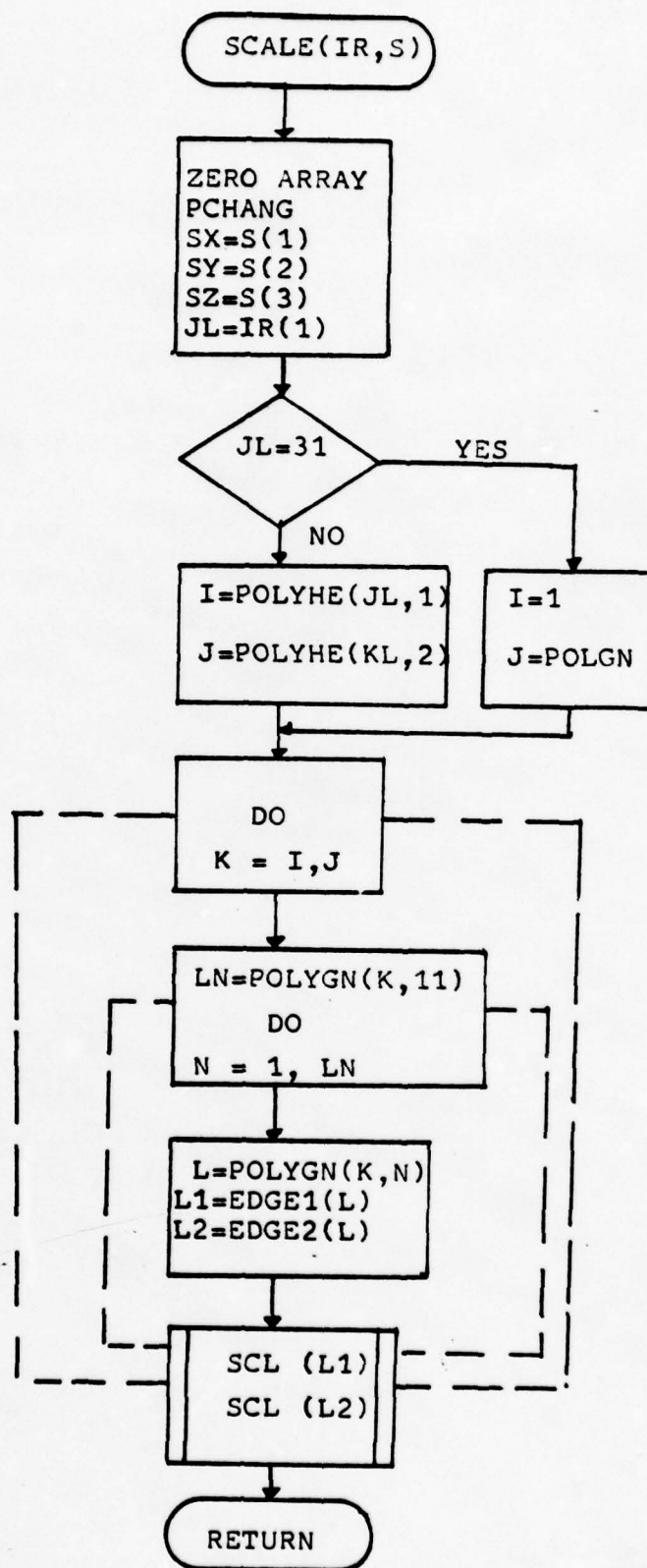
```
SUBROUTINE ROTZ(LR)
COMMON /AAA/ XF(120),YE(120),ZE(120),POINTN
COMMON /FF/ PCHANG,THETA
INTEGER PCHANG(200),POINTN
IF(PCHANG(LR).EQ.1) RETURN
XA=XF(LR)
YA=YE(LR)
YE(LR)=ROLL(YA,XA,THETA)
YA=-YA
XE(LR)=ROLL(XA,YA,THETA)
PCHANG(LR)=1
RETURN
END
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      SCALE: SCALES A SINGLE POLYHEDRON OR THE ENTIRE IMAGE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE SCALE(IR,S)
  DIMENSION S(3),IR(2)
  COMMON /AA/ POLYHE,POLYHN
  COMMON /AB/ POLYGN,POLGN,SHAD
  COMMON /AC/ EDGE1,EDGE2,EDGEN
  COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
  COMMON /FF/ PCHANG
  COMMON /HH/ SX,SY,SZ
  INTEGER POLYHE(10,2),POLYGN(60,11),EDGE1(100),EDGE2(100)
  &,POLYHN,POLGN,EDGEN,POINTN,SCALIT,SHAD(60),PCHANG(200)
  SX=S(1)
  SY=S(2)
  SZ=S(3)
  DO 30 I=1,POINTN
30 PCHANG(I)=0
  JL=IR(1)
  IF(JL.EQ.31) GO TO 132
  KL=IR(2)
  J=POLYHE(KL,2)
  II=POLYHE(JL,1)
  DO 133 K=II,J
    LN=POLYGN(K,11)
    DO 134 N=1,LN
      L=POLYGN(K,N)
      CALL SCL(EDGE1(L))
      CALL SCL(EDGE2(L))
134 CONTINUE
133 CONTINUE
  RETURN
132 DO 135 K=1,POINTN
    CALL SCL(K)
135 CONTINUE
  RETURN
  END

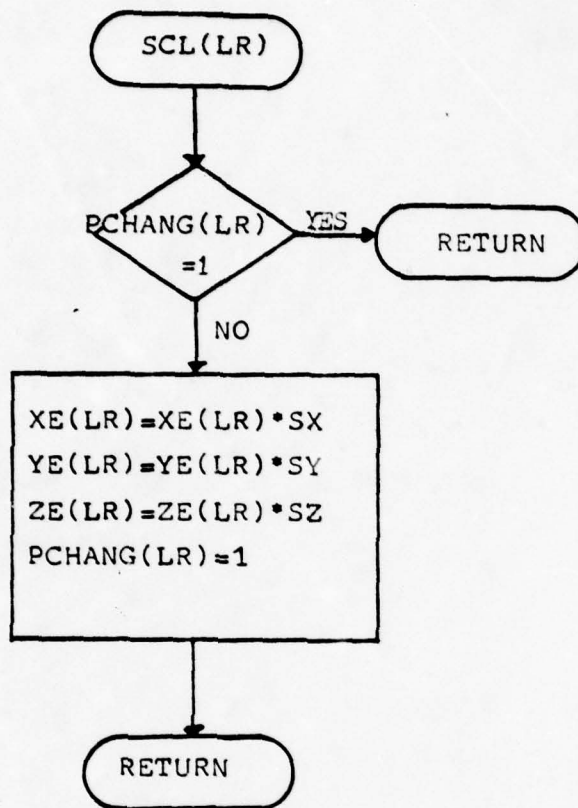
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      SCL: PERFORMS REPEATED MULT.'S TO SCALE THE SELECTED IMAGE
C      IN THE X, Y, AND Z DIRECTIONS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE SCL(LR)
COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
COMMON /HH/ SX,SY,SZ
COMMON /FF/ PCHANG,THETA
INTEGER PCHANG(200),POINTN
IF(PCHANG(LR).EQ.1) RETURN
XE(LR)=XE(LR)*SX
YE(LR)=YE(LR)*SY
ZE(LR)=ZE(LR)*SZ
PCHANG(LR)=1
RETURN
END

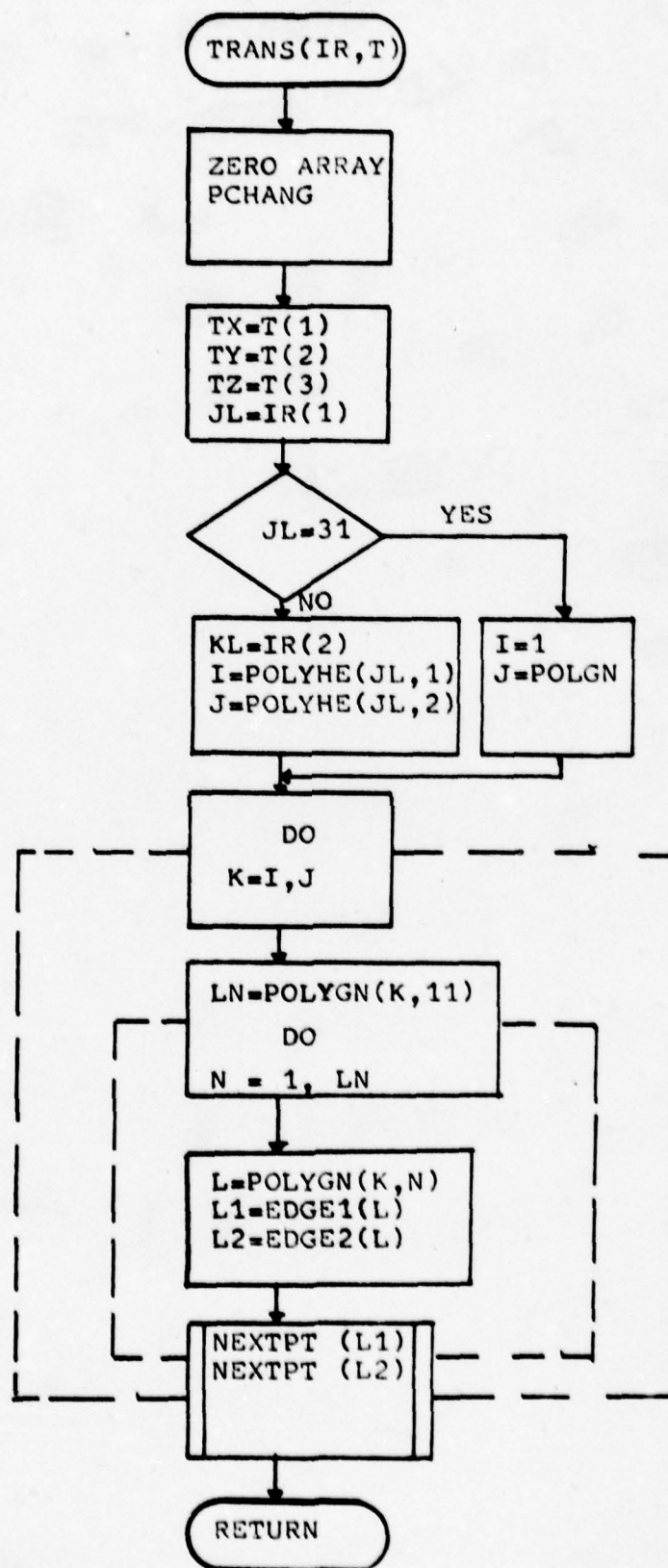
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   TRANSL: TRANSLATES A SINGLE POLYHEDRON OR THE ENTIRE IMAGE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE TRANSL(IR,I)
      DIMENSION T(3),IR(2)
      INTEGER POLYHE(10,2),POLYGN(60,11),EDGE1(100),EDGE2(100),
&PCHANG(200),POLGN,EDGEN,TRASTI,POINTN,POLYHN,SHAD(60)
      COMMON /AA/POLYHE,POLYHN
      COMMON /AH/POLYGN,POLGN,SHAD
      COMMON /AC/ EDGE1,EDGE2,EDGEN
      COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
      COMMON /FF/ PCHANG,IPETA
      COMMON /EE/IX,IY,IZ
      TX=T(1)
      TY=T(2)
      TZ=T(3)
      DO 30 II=1,POINTN
30  PCHANG(II)=0
      JL=IR(1)
      IF(JL.EQ.51) GO TO 103
      KL=IR(2)
      J=POLYHE(KL,2)
      II=POLYHE(JL,1)
      DO 105 K=1,J
          LN=POLYGN(K,11)
          DO 106 N=1,LN
              L=POLYGN(K,N)
              CALL NEXTPT(EDGE1(L))
              CALL NEXTPT(EDGE2(L))
106  CONTINUE
105  CONTINUE
      RETURN
103 DO 107 K=1,POINTN
          CALL NEXTPT(K)
107 CONTINUE
      RETURN
      END

```



CC

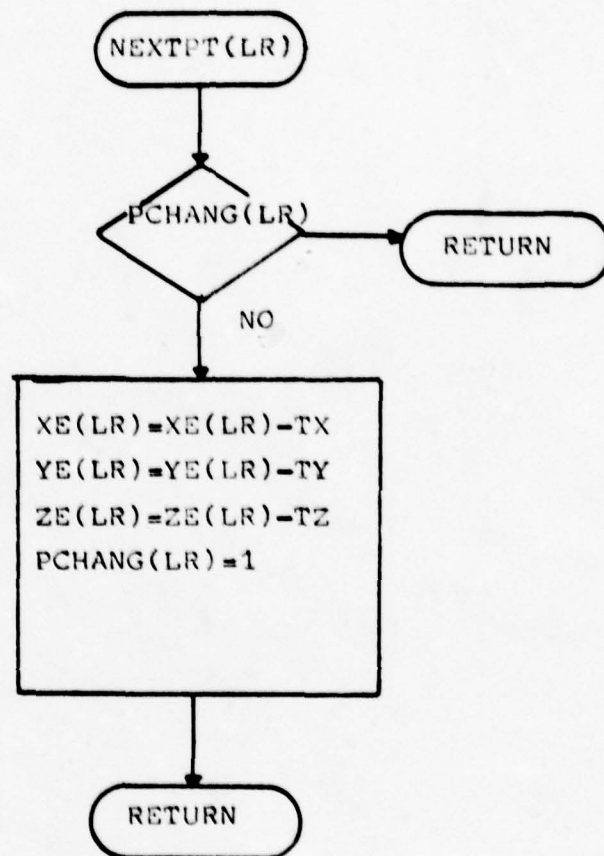
C

C     NEXTPT: PERFORMS REPEATED OPERATIONS FOR TRANSL

C

CC

```
SUBROUTINE NEXTPT(LR)
  INTEGER PCHANG(200),POININ
  COMMON /AAA/XE(120),YE(120),ZE(120),POININ
  COMMON /FF/PCHANG,THETA
  COMMON /FE/IX,TY,IZ
  IF(PCHANG(LR).EQ.1) RETURN
  XE(LR)=XE(LR)-TX
  YE(LR)=YE(LR)-TY
  ZE(LR)=ZE(LR)-IZ
  PCHANG(LR)=1
  RETURN
END
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      GMPRD: MULTIPLIES TWO 4 BY 4 MATRICES AND STORES THE RESULT
C      IN ANOTHER 4 BY 4 MATRIX.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE GMPRD(A,B,R,N,M,L)
3      DIMENSION A(4,4),B(4,4),R(4,4)
      DO 10 K=1,4
      DO 10 J=1,4
      R(K,J)=0.0
      DO 10 I=1,4
10     R(K,J)=R(K,J)+A(K,I)*B(I,J)
      RETURN
      END

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      GPRD: MULTIPLYS A 1 BY 4 VECTOR TIMES A 4 BY 4 MATRIX AND
C      STORES THE RESULT IN A 4 BY 1 VECTOR.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE GPRD(A,B,R)
      DIMENSION A(4),B(4,4),R(4)
      DO 10 I=1,4
      R(I)=0.0
      DO 10 J=1,4
10     R(I)=R(I)+A(J)*B(J,I)
      RETURN
      END

```

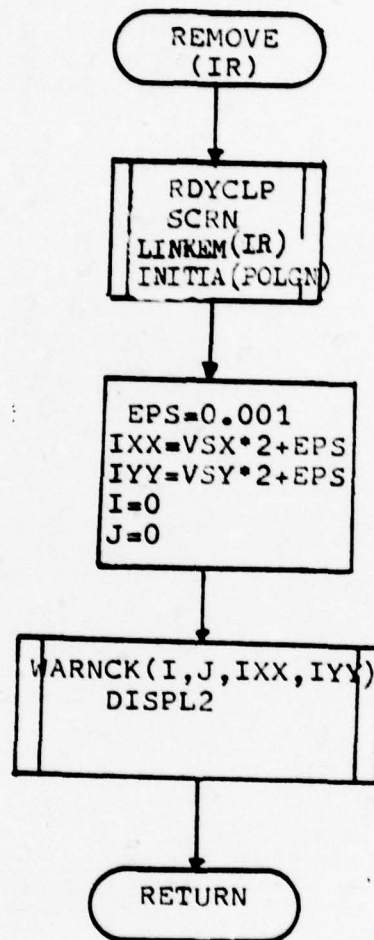
## 2. Hidden Line Removal

Included in this group were the subroutines utilized to remove hidden lines from the wire-frame figure produced by DISPLY. The subroutine REMOVE initiated this procedure when called.

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      REMOVE: MASTER SUBROUTINE WHICH DRAWS THE IMAGE, AFTER REMOVING
C      ALL HIDDEN LINES, ON THE SELECTED OUTPUT DEVICE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE REMOVE(IR)
      COMMON /AA/POLYHE(10,2),POLYHN
      DIMENSION IR(2)
      COMMON /AB/POLYGN,POLGN,SHAD
      COMMON /JJ/ V SX,V SY,V CX,V CY
      COMMON /DB/ JD
      INTEGER POLYGN(60,11),POLGN,SHAD(60),POLYHE,POLYHN
      CALL RDYCLP
      CALL SCRIN
      CALL LINKM(IR)
      CALL INITIA(POLGN)
      XX=VSX*2.+0.0001
      IXX=XX
      YY=VSY*2.+0.0001
      IYY=YY
      IPP=0
      JPP=0
      CALL WARNCK(IPP,JPP,IXX,IYY)
      JL=IR(1)
      IP=POLYHE(JL,1)
      ISHAD=SHAD(IP)
      CALL DISPL2(ISHAD)
      WRITE(6,1) JD
1  FORMAT(1X,'THE NUMBER OF STORAGE LOCATIONS NEEDED IS=',16)
      RETURN
      END

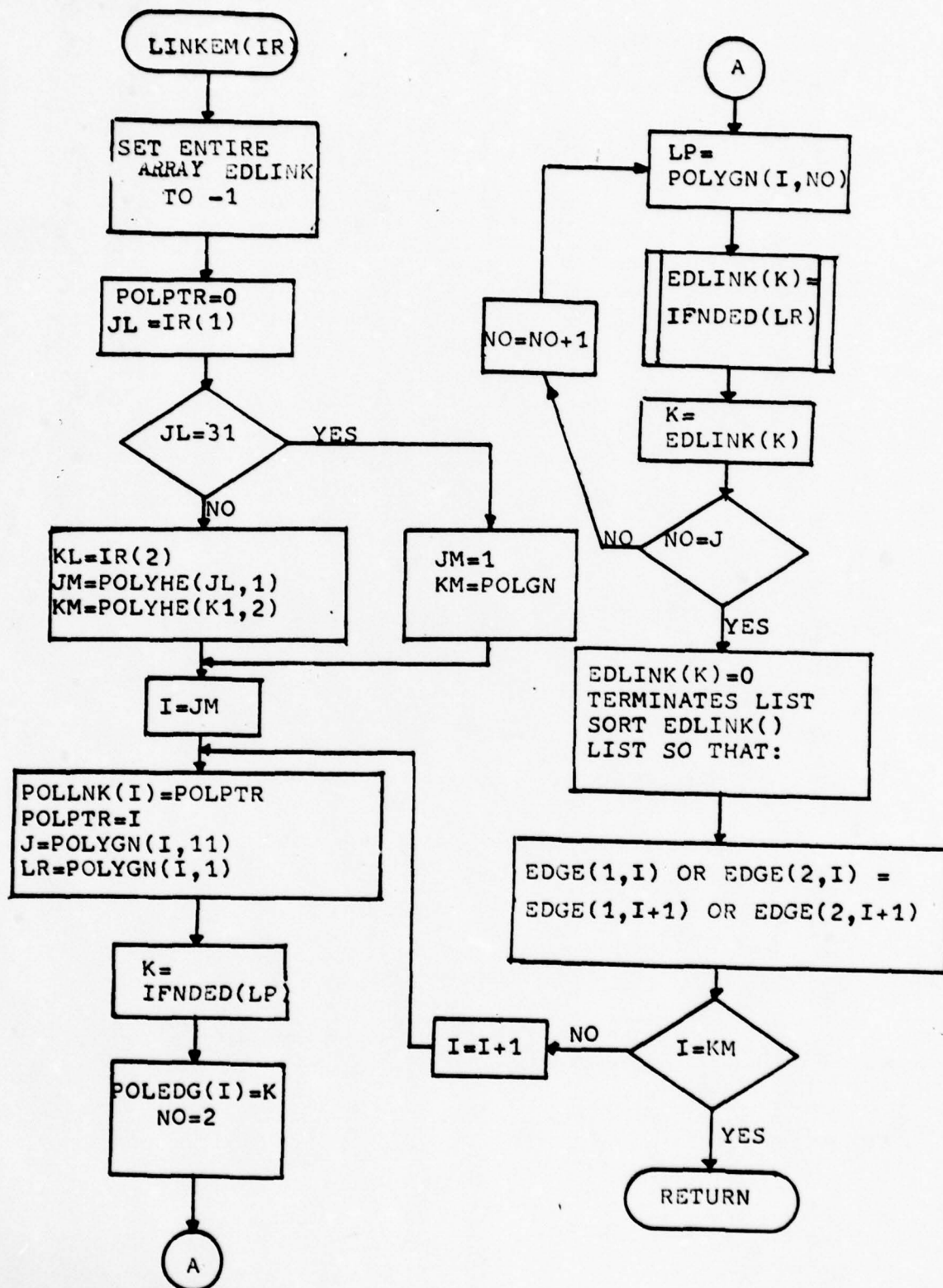
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   LINKEM: GENERATES LINKED LISTS FOR THE POLYGONS AND EDGES
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE LINKEM(IR)
  DIMENSION IR(2)
  COMMON /AA/ POLYHE, POLYHN
  COMMON /AB/ POLYGN, POLGN, SHAD
  COMMON /AAD/ EDGE, EDGEN
  COMMON /RA/ EDLINK, POLEDG, NEXTE
  COMMON /RB/ POLLNK, POLPTR
  INTEGER POLYGN(60,11), EDGE(2,200), EDLINK(200), POLEDG(60)
  &, POLGN, EDGEN, POLPTR, SHAD(60), POLYHE(10,2), POLYHN, POLLNK(60)
  DO 31 I=1,200
31 EDLINK(I)=-1
  POLPTR=0
  JL=IR(1)
  IF(JL.EQ.51) GO TO 2000
  KL=IR(2)
  JM=POLYHE(JL,1)
  KM=POLYHE(KL,2)
  GO TO 2020
2000 JM=1
  KM=POLGN
2020 DO 200 I=JM,KM
  POLLNK(I)=POLPTR
  POLPTR=I
  J=POLYGN(I,11)
  LR=POLYGN(I,1)
  K=IFNDED(LR)
  POLEDG(I)=K
  DO 201 NO=2,J
    LR=POLYGN(I,NO)
    EDLINK(K)=IFNDED(LR)
    K=EDLINK(K)
201 CONTINUE
  EDLINK(K)=0
  K=POLEDG(I)
  J=EDGE(2,K)
  JT=EDLINK(K)
  IF(J.NE.EDGE(1,JT).AND.J.NE.EDGE(2,JT)) CALL ISWAP
  & (EDGE(1,K),EDGE(2,K))
202 IF(K.EQ.0) GO TO 200
  J=EDLINK(K)
  IF(J.NE.0.AND.EDGE(2,K).NE.EDGE(1,J)) CALL
  & ISWAP(EDGE(1,J),EDGE(2,J))
  K=J
  GO TO 202
200 CONTINUE
  RETURN
  END

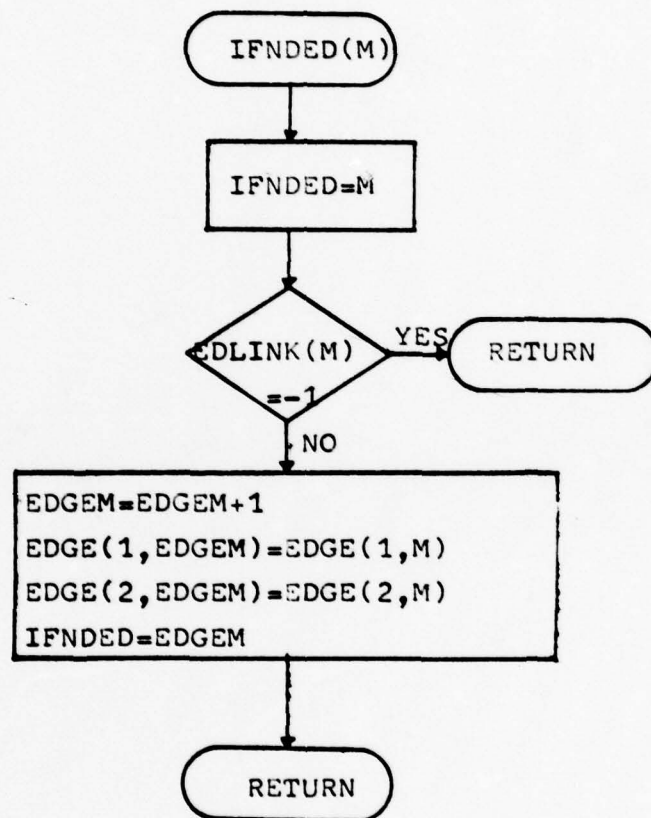
```



```

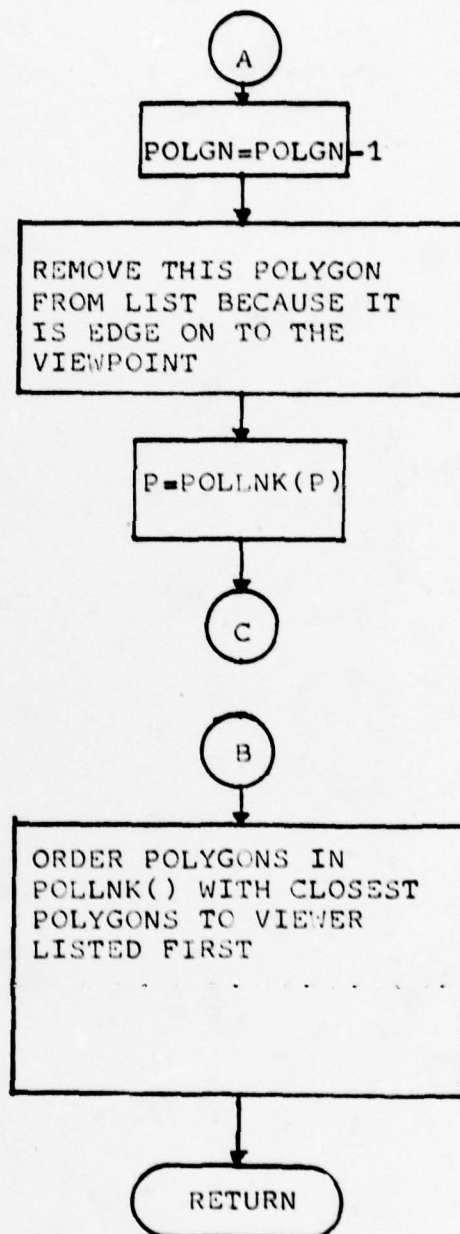
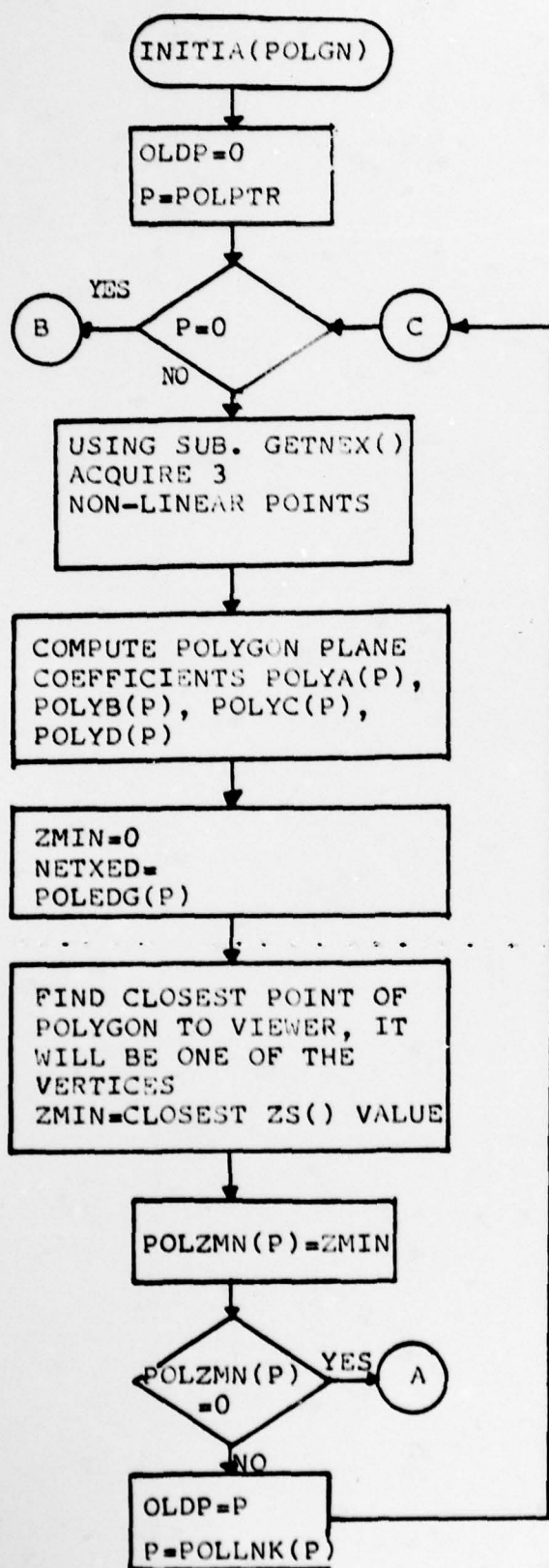
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      IFNDED: USED BY LINKEM TO FIND EMPTY STORAGE LOCATIONS IN
C      THE LIST USED TO LINK THE EDGES OF EACH POLYGON.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION IFNDED(M)
      COMMON /AAD/ EDGF,EDGEM
      COMMON /RA/ EDLINK,POLEDG,NEXTED
      INTEGER EDGE(2,200),EDGEM,EDLINK(200),POLEDG(60)
      IFNDED=M
      IF(EDLINK(M).EQ.-1) RETURN
      EDGEM=EDGEM+1
      EDGE(1,EDGEM)=EDGE(1,M)
      EDGE(2,EDGEM)=EDGE(2,M)
      IFNDED=EDGEM
      RETURN
      END

```





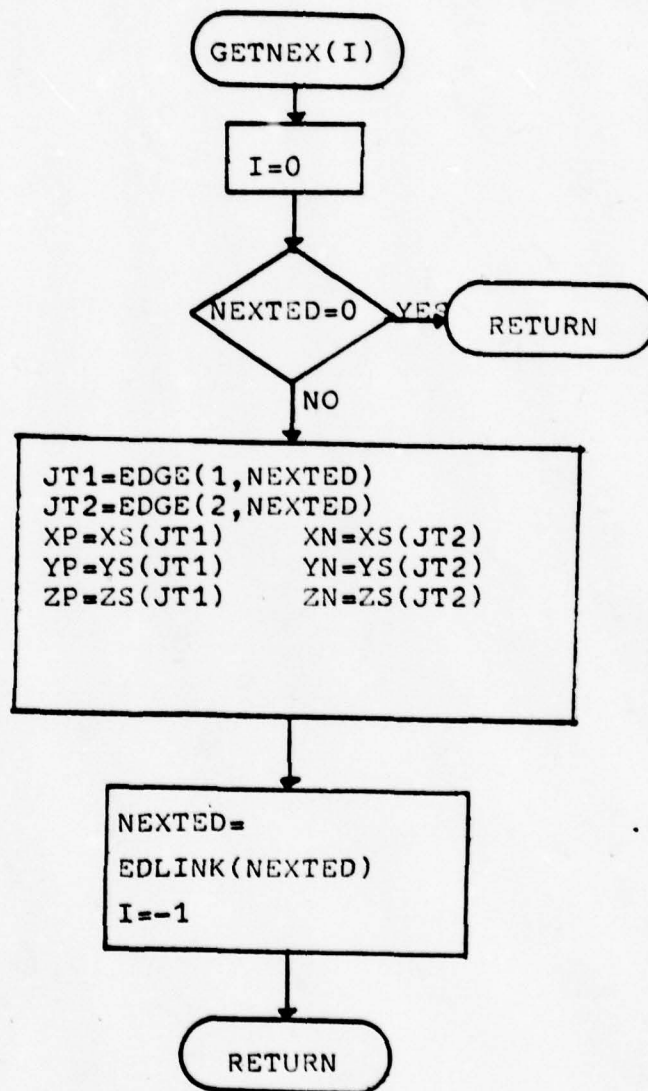
```
      OLDP=J  
      GO TO 215  
213  POLPTR=J  
      GO TO 214  
212  OLDP=P  
      P=POLLNK(P)  
      GO TO 215  
211  IF(CHANGE.EQ.1) GO TO 203  
      RETURN  
      END
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   GETNEX: GETS THE INDEX OF THE NEXT EDGE FOR THE CURRENT POLYGON
C   AND THEN FINDS THE TWO END POINTS OF THIS EDGE TO PASS TO THE
C   CALLING SUBROUTINE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE GETNEX(I)
COMMON /AAB/ EDGE,EDGEN
COMMON /AAB/ XS(120),YS(120),ZS(120),POINTM
COMMON /RA/ EDLINK,POLEDG,NEXTED
COMMON /RD/ XP,YP,ZP,XN,YN,ZN
COMMON /DC/ JT1,JT2
INTEGER EDGE(2,200),EDLINK(200),POLEDG(60),EDGEN,POINTM
I=0
IF(NEXTED.EQ.0) RETURN
JT1=EDGE(1,NEXTED)
XP=XS(JT1)
YP=YS(JT1)
ZP=ZS(JT1)
JT2=EDGE(2,NEXTED)
XN=XS(JT2)
YN=YS(JT2)
ZN=ZS(JT2)
NEXTED=EDLINK(NEXTED)
I=-1
RETURN
END

```





```

ZMIN4=0.
ZMAX1=0.
ZMAX2=0.
ZMAX3=0.
ZMAX4=0.
HIDER=0
PENET=0
227 IF (SURRND.EQ.0) GO TO 224
Z1=GETZ(SURRND,ALX,ARY)
Z2=GETZ(SURRND,ALX,ATY)
Z3=GETZ(SURRND,ARX,ARY)
Z4=GETZ(SURRND,ARX,ATY)
IF (Z1.GE.ZMIN1) GO TO 225
IF (Z2.GE.ZMIN2) GO TO 225
IF (Z3.GE.ZMIN3) GO TO 225
IF (Z4.GE.ZMIN4) GO TO 225
HIDER=SURRND
ZMIN1=Z1
ZMAX1=Z1
ZMIN2=Z2
ZMAX2=Z2
ZMIN3=Z3
ZMAX3=Z3
ZMIN4=Z4
ZMAX4=Z4
PENET=0
GO TO 226
225 IF (Z1.LE.ZMAX1) GO TO 292
IF (Z2.LE.ZMAX2) GO TO 292
IF (Z3.LE.ZMAX3) GO TO 292
IF (Z4.LE.ZMAX4) GO TO 292
226 SURRND=POLLST(SURRND)
GO TO 227
292 PENET=1
IF (Z1.LT.ZMIN1) ZMIN1=Z1
IF (Z1.GT.ZMAX1) ZMAX1=Z1
IF (Z2.LT.ZMIN2) ZMIN2=Z2
IF (Z2.GT.ZMAX2) ZMAX2=Z2
IF (Z3.LT.ZMIN3) ZMIN3=Z3
IF (Z3.GT.ZMAX3) ZMAX3=Z3
IF (Z4.LT.ZMIN4) ZMIN4=Z4
IF (Z4.GT.ZMAX4) ZMAX4=Z4
GO TO 226
224 IF (PENET.EQ.1) GO TO 226
OLDP=0
P=INTER
2244 IF (P.EQ.0) GO TO 229
IF (HIDER.EQ.0) GO TO 229
Z1=GETZ(P,ALX,ARY)
Z2=GETZ(P,ALX,ATY)
Z3=GETZ(P,ARX,ARY)
Z4=GETZ(P,ARX,ATY)
IF (Z1.LE.ZMAX1) GO TO 230
IF (Z2.LE.ZMAX2) GO TO 230
IF (Z3.LE.ZMAX3) GO TO 230
IF (Z4.LE.ZMAX4) GO TO 230
J=POLLST(P)
IF (OLDP.EQ.0) GO TO 2311
POLLST(OLDP)=J
GO TO 231
230 OLDP=P
PENET=1
IF (Z1.GE.ZMIN1) GO TO 229

```

```

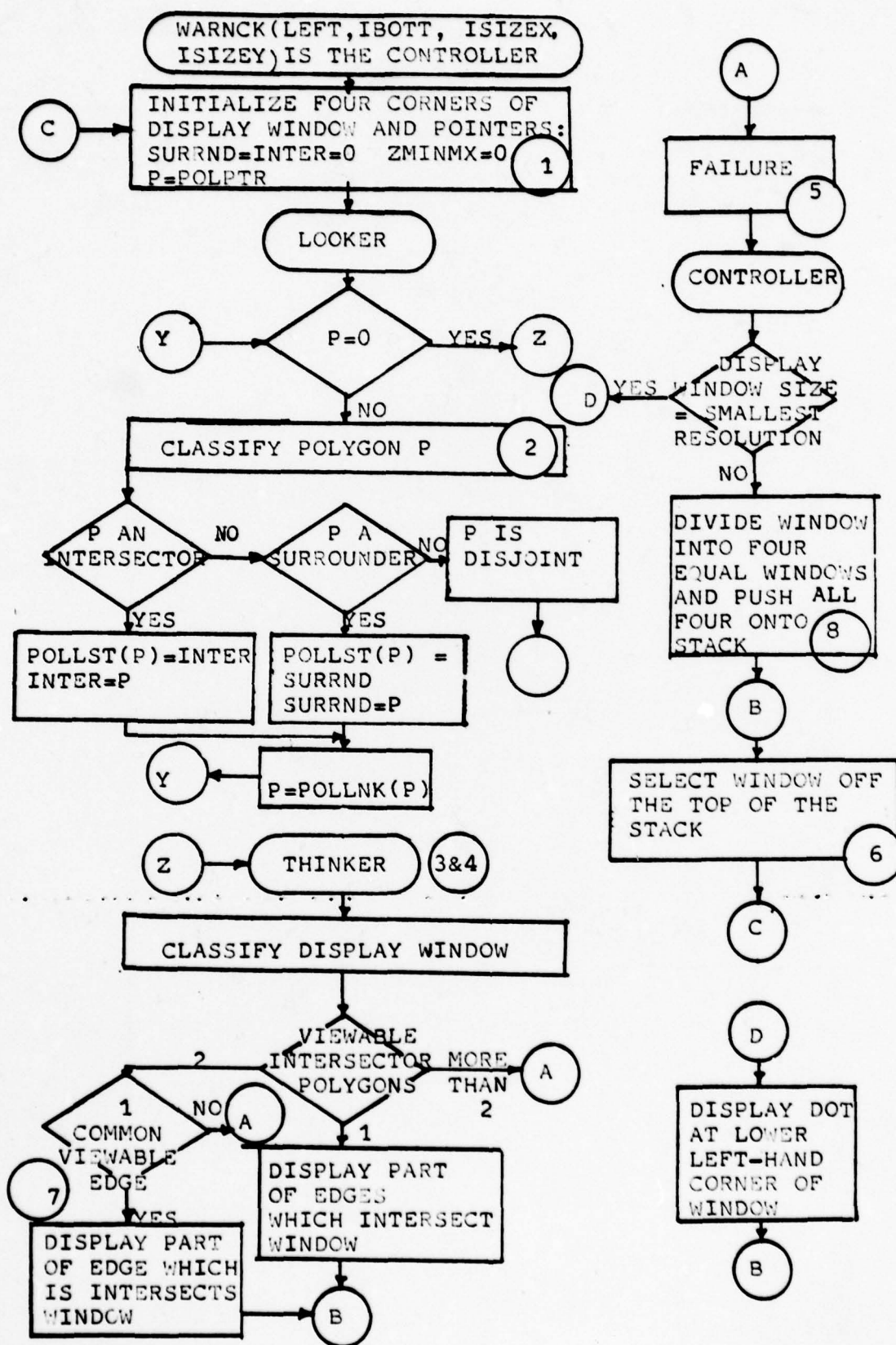
IF(Z2.GE.ZMIN2) GO TO 228
IF(Z3.GE.ZMIN3) GO TO 228
IF(Z4.GE.ZMIN4) GO TO 228
PENET=0
GO TO 231
2311 INTER=J
231 P=POLLST(P)
GO TO 2244
229 IF(INTER.EQ.0) GO TO 232
IF(POLLST(INTER).NE.0) GO TO 228
NEXTED=POLEDG(INTER)
233 CALL GETNEX(I)
IF(I.EQ.0) GO TO 232
CALL CLIP2(JR)
IF(JR.EQ.-1) CALL SHOWIT(XX(1),YY(1),XX(2),YY(2),ISHAD,0)
GO TO 233
2281 IF(PENET.EQ.1) GO TO 234
LP=POLLST(INTER)
IF(POLLST(LP).NE.0) GO TO 234
NEXTED=POLEDG(INTER)
IAT=0
2331 CALL GETNEX(I)
IF(I.EQ.0) GO TO 2282
CALL CLIP2(JR)
IF(JR.EQ.0) GO TO 2331
IAT=IAT+1
JJ1=J11
JJ2=J12
IF(JJ1.GT.JJ2) CALL ISWAP(JJ1,JJ2)
X11=XX(1)
Y11=YY(1)
X12=XX(2)
Y12=YY(2)
GO TO 2331
2282 IF(IAT.GT.1) GO TO 234
IAT=0
NEXTED=POLEDG(LP)
2332 CALL GETNEX(I)
IF(I.EQ.0) GO TO 2283
CALL CLIP2(JR)
IF(JR.EQ.0) GO TO 2332
IAT=IAT+1
JK1=J11
JK2=J12
IF(JK1.GT.JK2) CALL ISWAP(JK1,JK2)
GO TO 2332
2283 IF(IAT.GT.1) GO TO 234
IF(JK1.NE.JJ1) GO TO 234
IF(JK2.NE.JJ2) GO TO 234
CALL SHOWIT(X11,Y11,X12,Y12,ISHAD,0)
GO TO 232
228 IF(ISIZEX.GT.1) GO TO 2281
IF(ISIZEY.GT.1) GO TO 2281
CALL SHOWIT(RLY,RBY,RLX,RBY,ISHAD,1)
GO TO 232
234 ICC=ISIZEX/2
ICX=ISIZEX-ICC*2
ICC=ISIZEY/2
ICY=ISIZEY-ICC*2
IF(ISIZEX.EQ.1) GO TO 350
ISIZEX=ISIZEX/2
IF(ISIZEY.EQ.1) GO TO 360
ISIZEY=ISIZEY/2

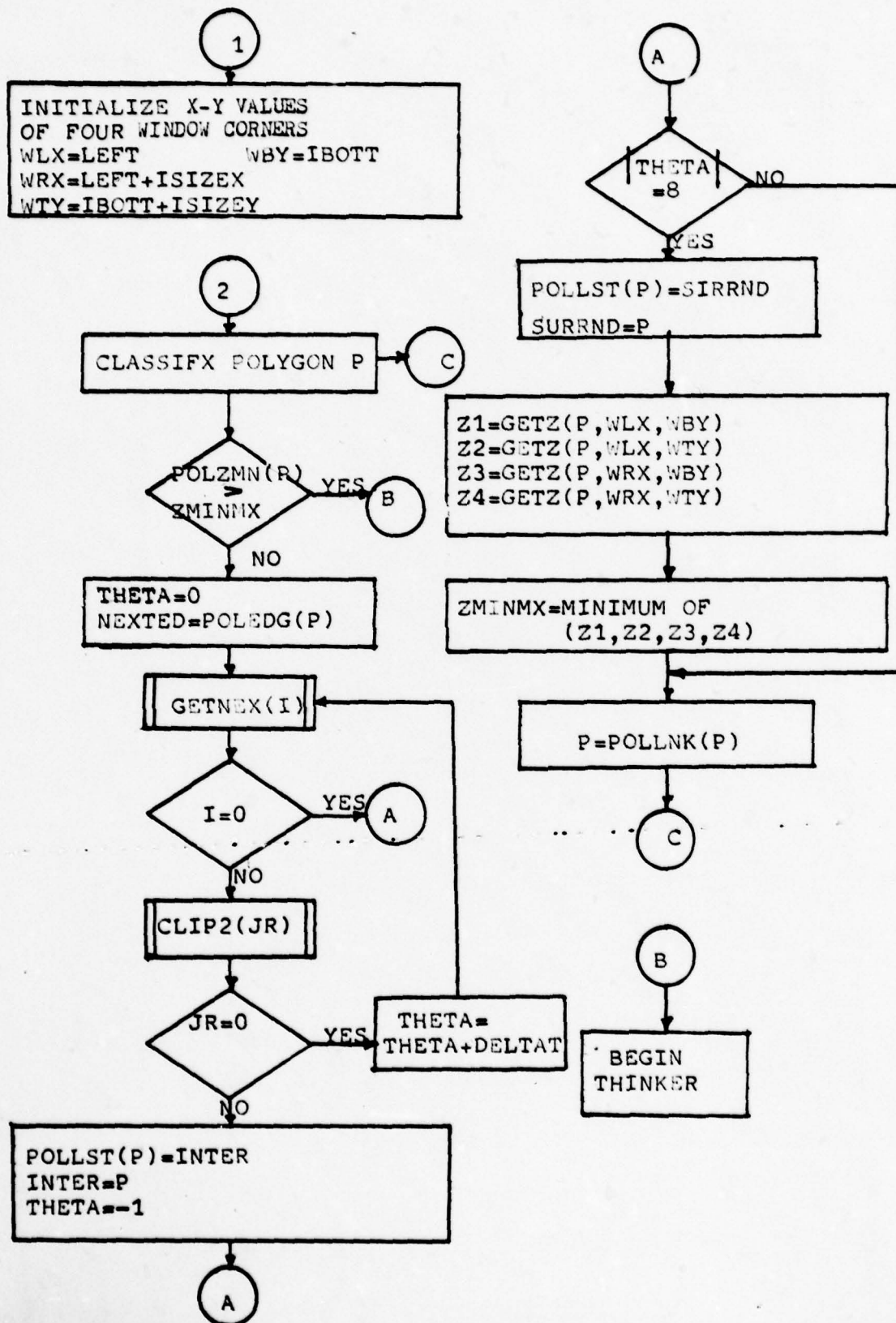
```

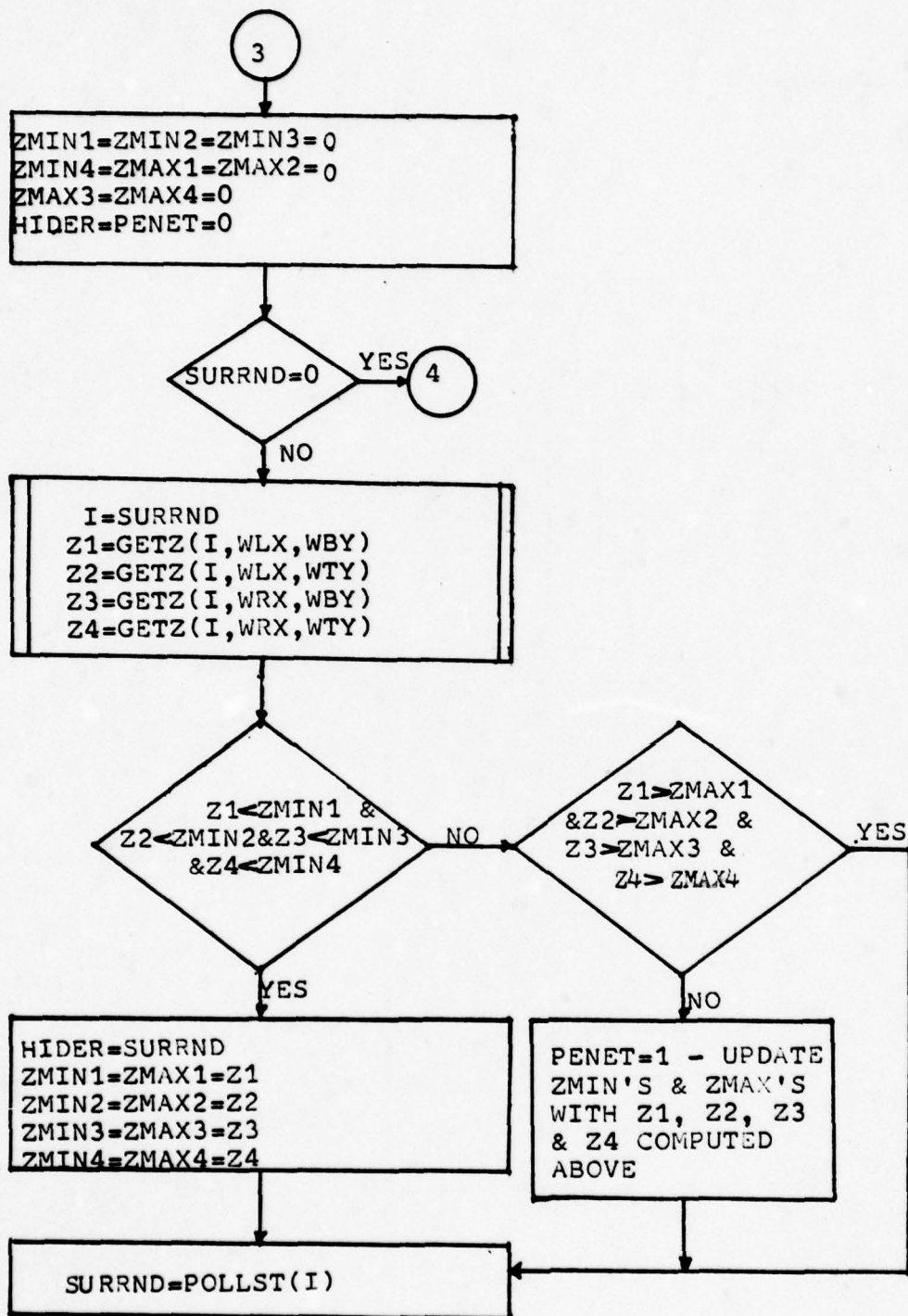
```

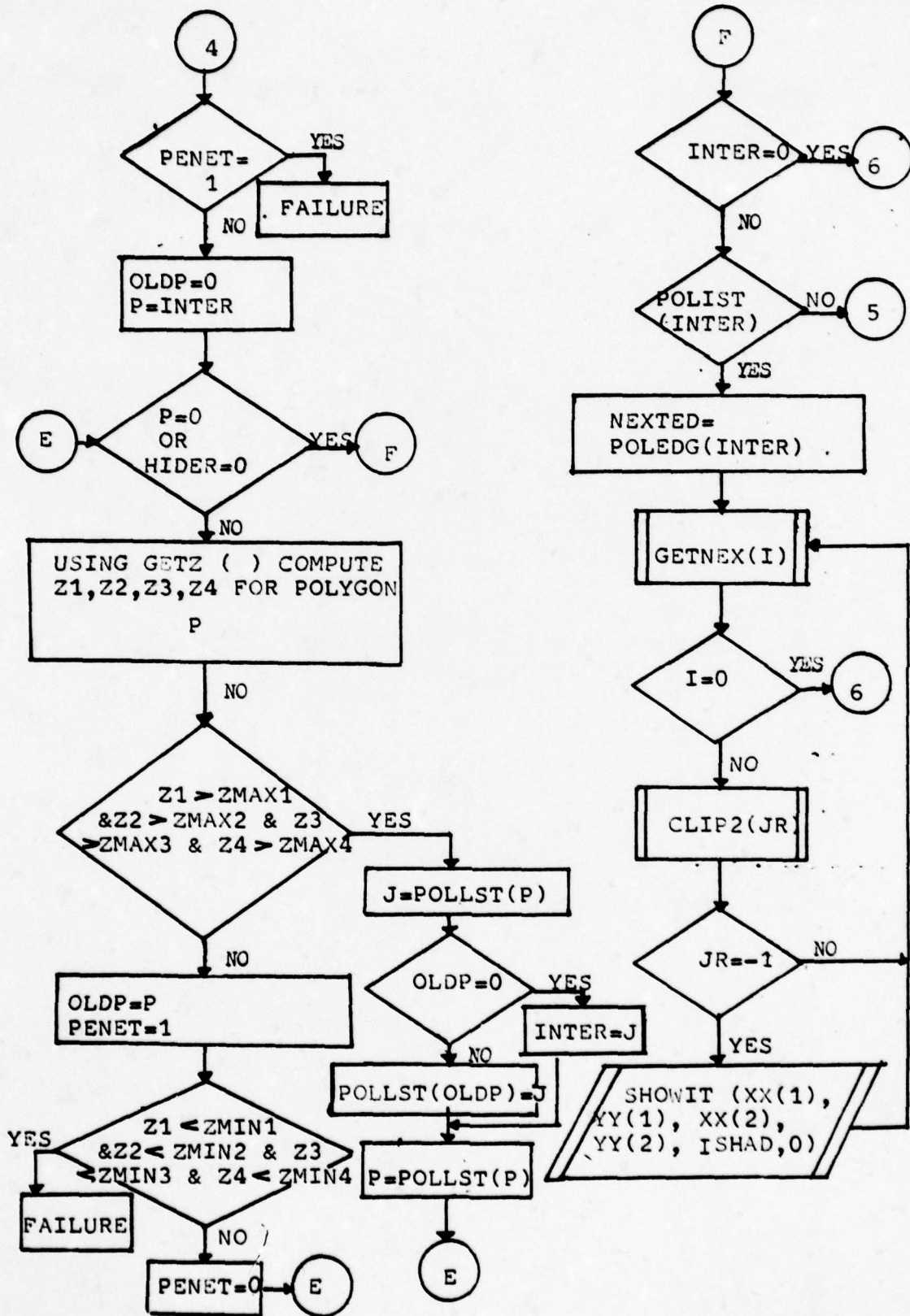
LEFT=LEFT+ISIZEX
ISIZEX=ISIZEX+ICX
DO 235 I=1,4
    ISTPTR=ISTPTR+1
    ISTACK(ISTPTR,1)=LEFT
    ISTACK(ISTPTR,2)=IBOTT
    ISTACK(ISTPTR,3)=ISIZEA
    ISTACK(ISTPTR,4)=ISIZEY
    GO TO (236,237,238,235),I
236 IBOTT=IBOTT+ISIZEY
    ISIZEY=ISIZEY+ICY
    GO TO 235
237 ISIZEX=ISIZEA-ICX
    LEFT=LEFT-ISIZEX
    GO TO 235
238 ISIZEY=ISIZEY-ICY
    IBOTT=IBOTT-ISIZEY
235 CONTINUE
232 IF(ISTPTR.LE.0) RETURN
    LEFT=ISTACK(ISTPTR,1)
    IBOTT=ISTACK(ISTPTR,2)
    ISIZEX=ISTACK(ISTPTR,3)
    ISIZEY=ISTACK(ISTPTR,4)
    ISTPTR=ISTPTR-1
    GO TO 239
350 ISIZEY=ISIZEY/2
    IBOTT=IBOTT+ISIZEY
    ISIZEY=ISIZEY+ICY
    DO 351 I=1,2
        ISTPTR=ISTPTR+1
        ISTACK(ISTPTR,1)=LEFT
        ISTACK(ISTPTR,2)=IBOTT
        ISTACK(ISTPTR,3)=ISIZEX
        ISTACK(ISTPTR,4)=ISIZEY
        ISIZEY=ISIZEY-ICY
        IBOTT=IBOTT-ISIZEY
351 CONTINUE
    GO TO 232
360 LEFT=LEFT+ISIZEX
    ISIZEX=ISIZEX+ICX
    DO 361 I=1,2
        ISTPTR=ISTPTR+1
        ISTACK(ISTPTR,1)=LEFT
        ISTACK(ISTPTR,2)=IBOTT
        ISTACK(ISTPTR,3)=ISIZEX
        ISTACK(ISTPTR,4)=ISIZEY
        ISIZEX=ISIZEX-ICX
        LEFT=LEFT-ISIZEX
361 CONTINUE
    GO TO 232
END

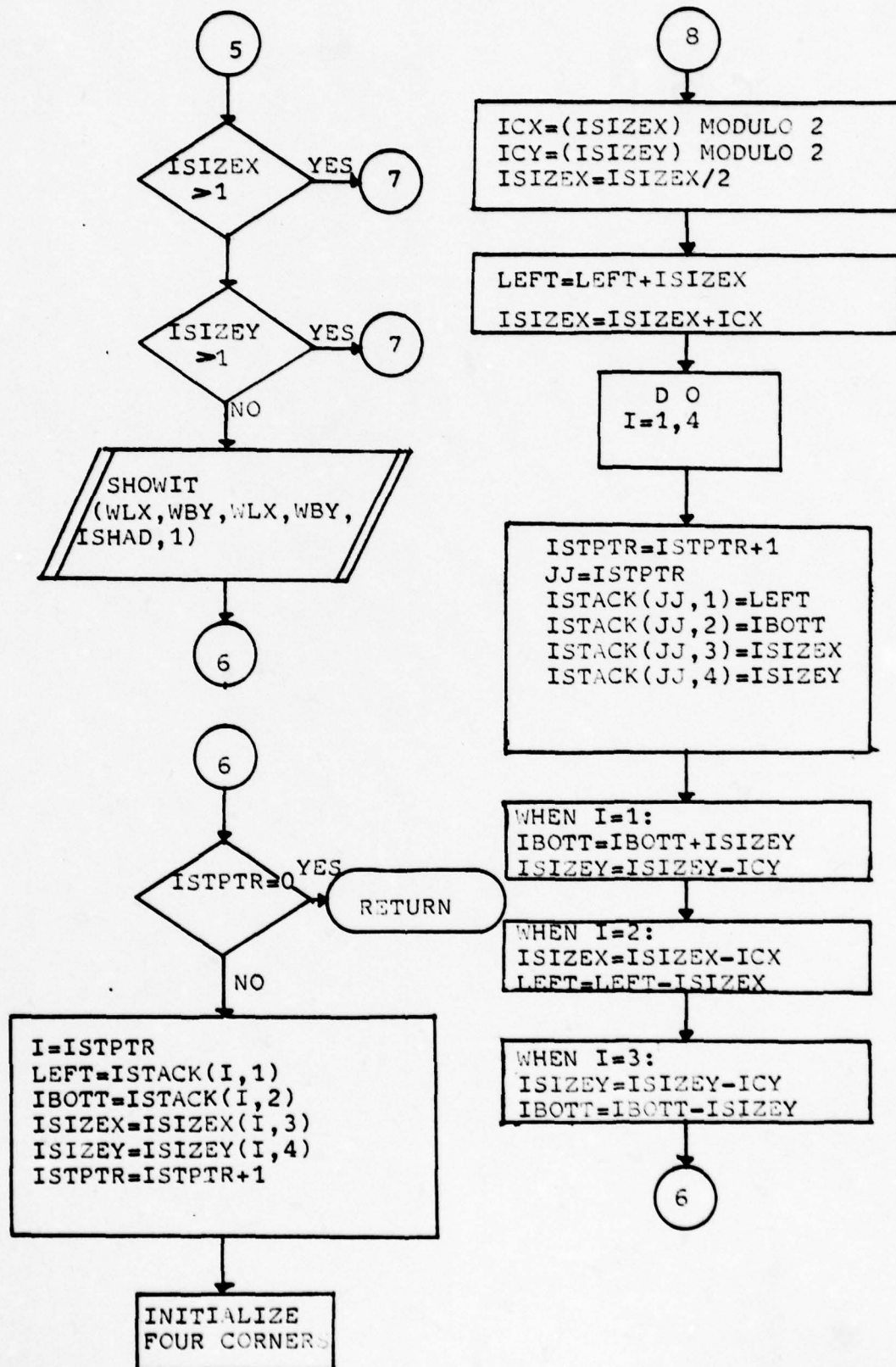
```

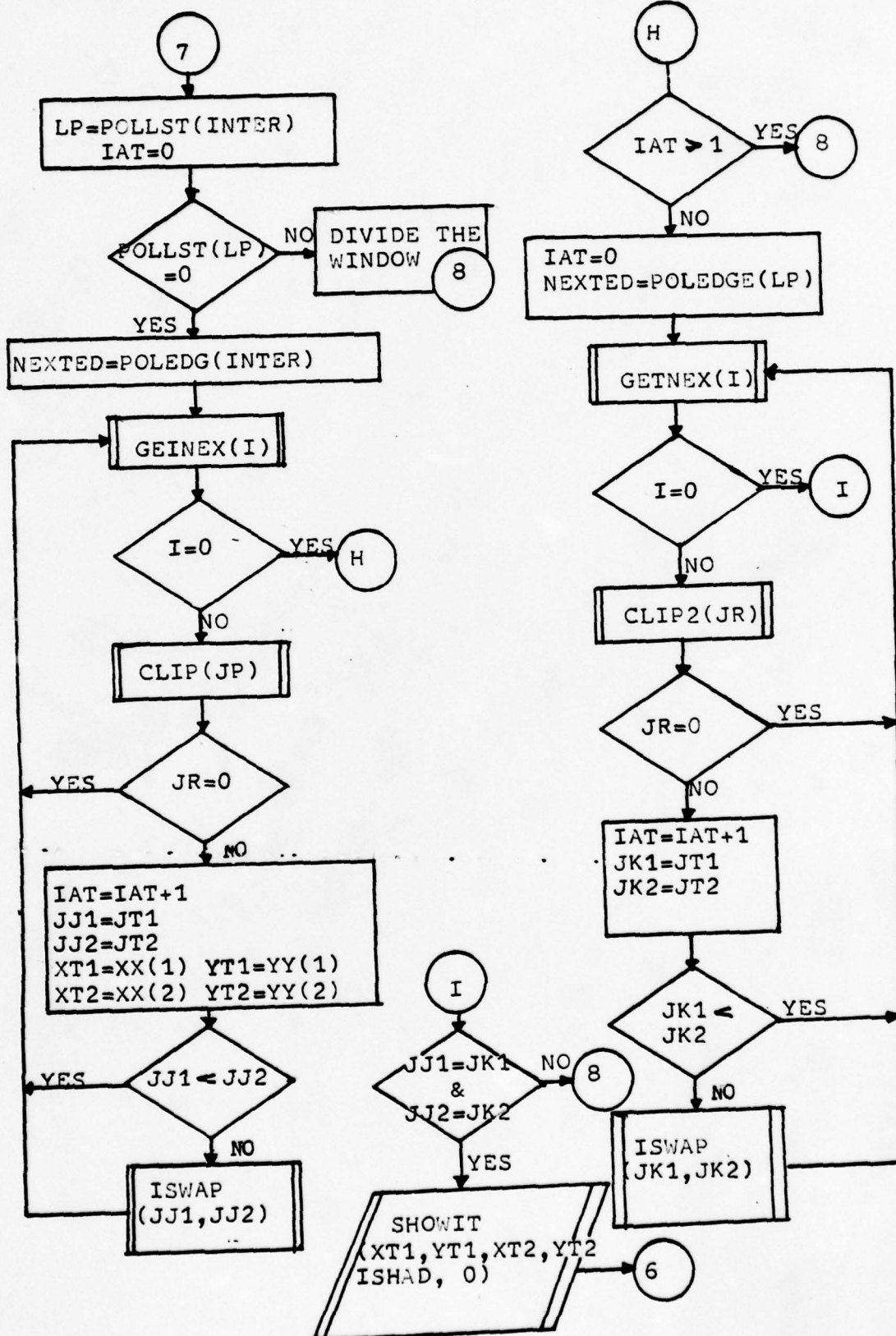








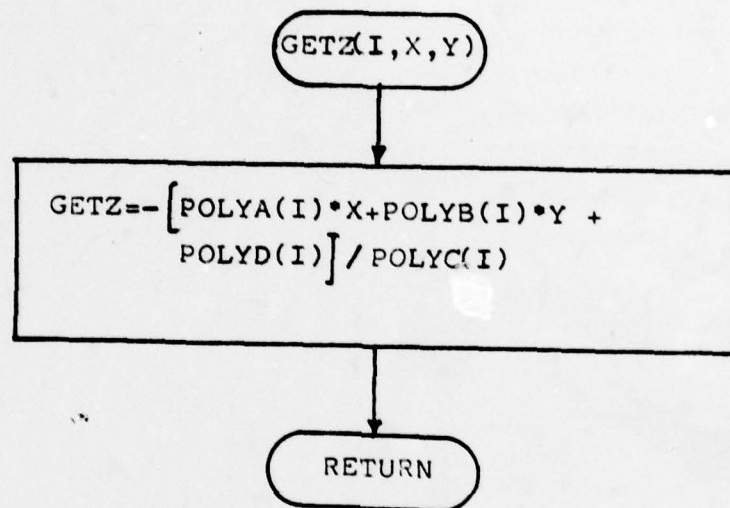




```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   GETZ: DETERMINES THE DEPTH OF THE SELECTED POLYGON AT ANY POINT
C   ON THE DISPLAY SCREEN, I.E. THE SCREEN COORDINATE Z VALUE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
FUNCTION GETZ(I,X,Y)
COMMON /RC/ POLYA(60),POLYB(60),POLYC(60),POLYD(60),POLZMN(60)
GETZ=(-POLYA(I)*X-POLYB(I)*Y-POLYD(I))/POLYC(I)
RETURN
END

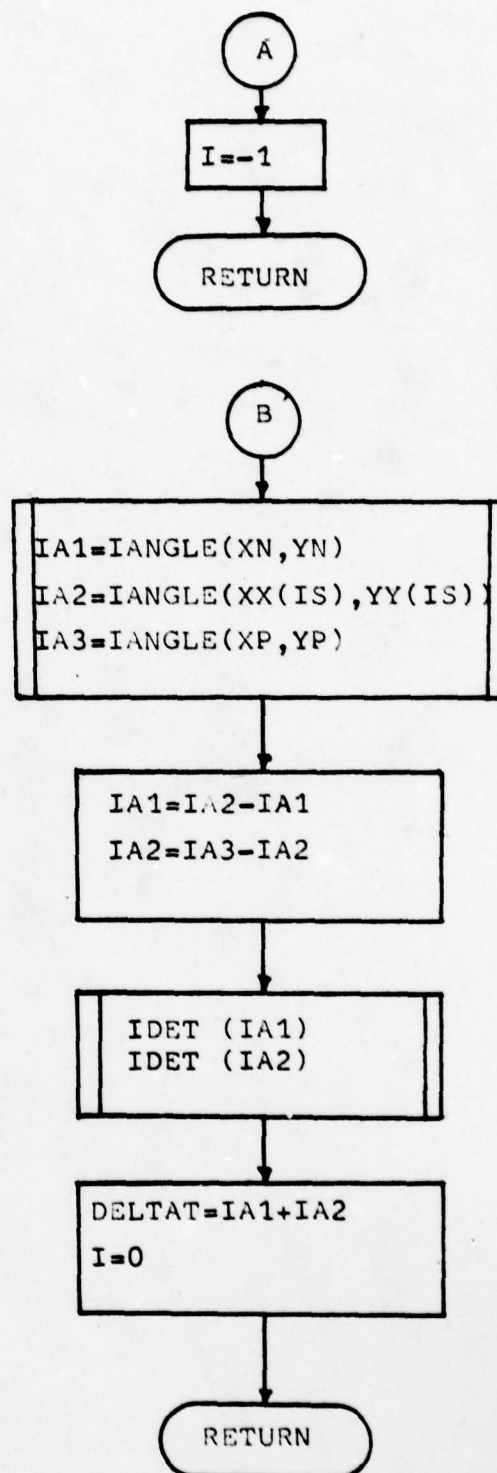
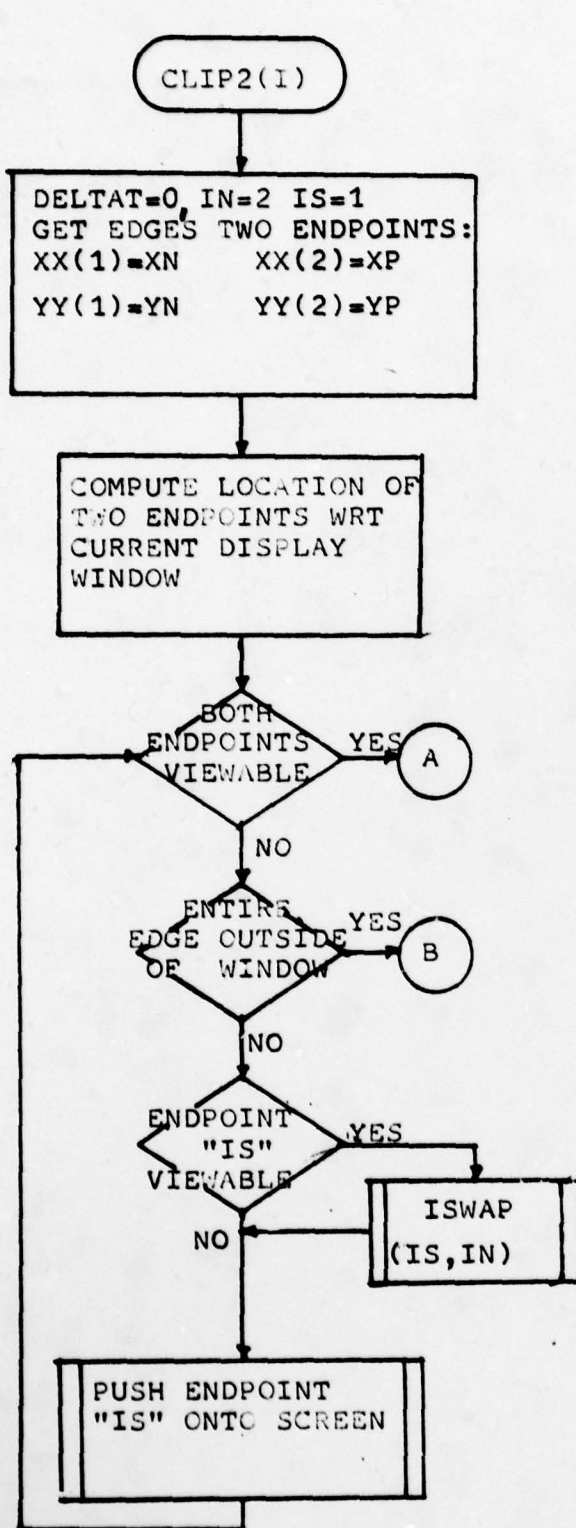
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      CLIP2: CLIPS THE SELECTED EDGE AGAINST THE CURRENT DISPLAY
C      WINDOW IF THE EDGE INTERSECTS THE WINDOW OR COMPUTES THE
C      ANGLE SUBTENDED BY THE EDGE WHICH IS USED TO DETERMINE IF THE
C      CURRENT POLYGON SURROUNDS THIS WINDOW.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE CLIP2(I)
COMMON /RG/ DELTA1,WLX,WRX,WRY,WTY
COMMON /RD/XP,YP,ZP,XN,YN,ZN
COMMON /RH/ XX(2),YY(2)
COMMON /R1/ ICHK(2,4)
COMMON /RJ/ IC(2),IS,IN
INTEGER DELTA1
DELTA1=0
IN=2
IS=1
XX(1)=XN
XX(2)=XP
YY(1)=YN
YY(2)=YP
IC(1)=JCODE(XX(1),YY(1),1)
IC(2)=JCODE(XX(2),YY(2),2)
257 ICOT=IC(1)+IC(2)
IF(ICOT.EQ.0) GO TO 250
DO 55 KK=1,4
IAAA=IC(1,KK)+IC(2,KK)
IF(IAAA.EQ.2) GO TO 251
55 CONTINUE
IF(IC(IS).EQ.0) CALL ISWAP(IS,IN)
IF(IC(1,IS).EQ.0) GO TO 253
CALL PUSH(0,WLX)
GO TO 257
253 IF(IC(2,IS).EQ.0) GO TO 254
CALL PUSH(0,WRX)
GO TO 257
254 IF(IC(3,IS).EQ.0) GO TO 255
CALL PUSH(1,WRY)
GO TO 257
255 CALL PUSH(1,WTY)
GO TO 257
250 I=-1
RETURN
251 IA1=IANGLE(XN,YN)
IA2=IANGLE(XX(IS),YY(IS))
IA3=IANGLE(XP,YP)
IA1=IA2-IA1
IA2=IA3-IA2
CALL IDET(IA1)
CALL IDET(IA2)
DELTA1=IA1+IA2
I=0
RETURN
END

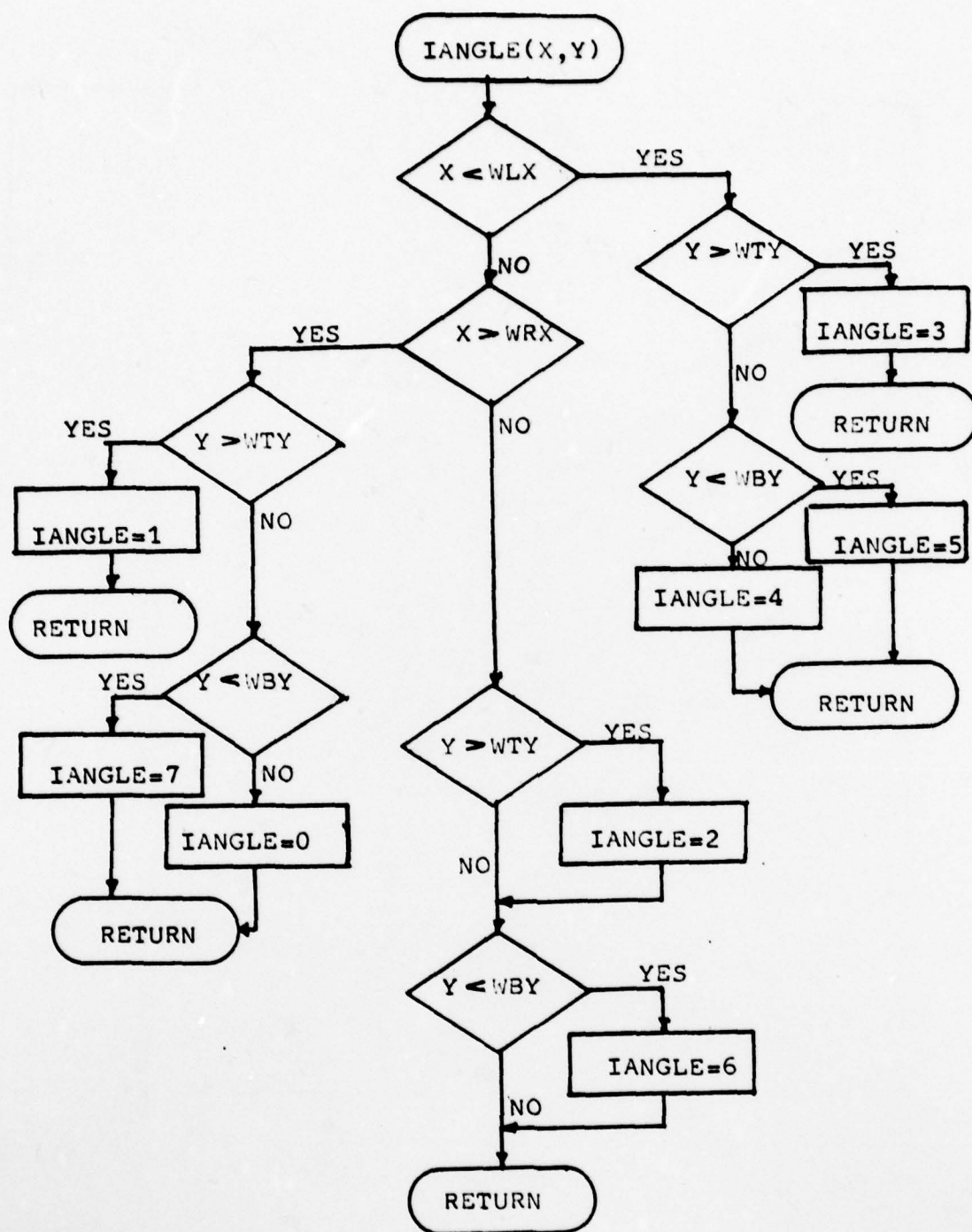
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      IANGLE: USED BY CLIP2 TO DETERMINE THE ANGLE SUBTENDED BY AN EDGE
C      WHICH DOES NOT INTERSECT THE CURRENT WINDOW.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION IANGLE(X,Y)
      COMMON /RG/ DELTAT,WLX,WRX,WBY,WTY
      INTEGER DELTAT
      IF(X.GE.WLX) GO TO 280
      IF(Y.LE.WTY) GO TO 281
      IANGLE=3
      RETURN
281 IF(Y.GE.WBY) GO TO 282
      IANGLE=5
      RETURN
282 IANGLE=4
      RETURN
280 IF(X.LE.WRX) GO TO 283
      IF(Y.LE.WTY) GO TO 284
      IANGLE=1
      RETURN
284 IF(Y.GE.WBY) GO TO 285
      IANGLE=7
      RETURN
285 IANGLE=0
      RETURN
283 IF(Y.GT.WTY) IANGLE=2
      IF(Y.LT.WBY) IANGLE=6
      RETURN
      END

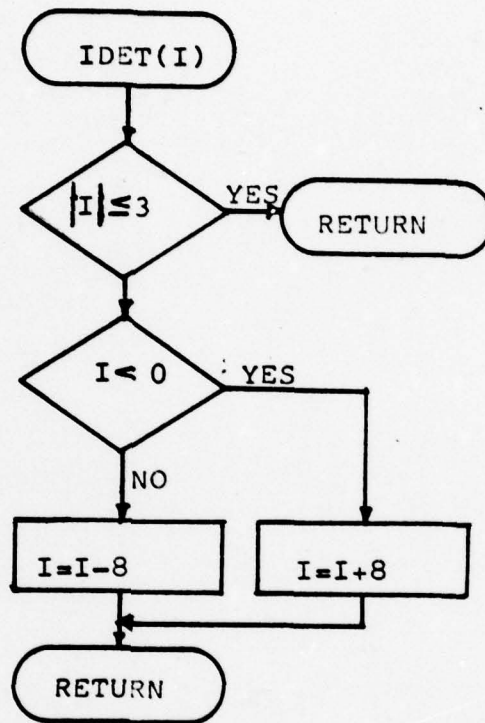
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      IDET: USED BY CLIP2 TO CORRECT THE SUBTENDED ANGLE OF AN EDGE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE IDET(I)
      IF (IABS(I).LE.3) RETURN
      IF (I.GE.0) GO TO 260
      I=I+8
      RETURN
260 I=I-8
      RETURN
      END

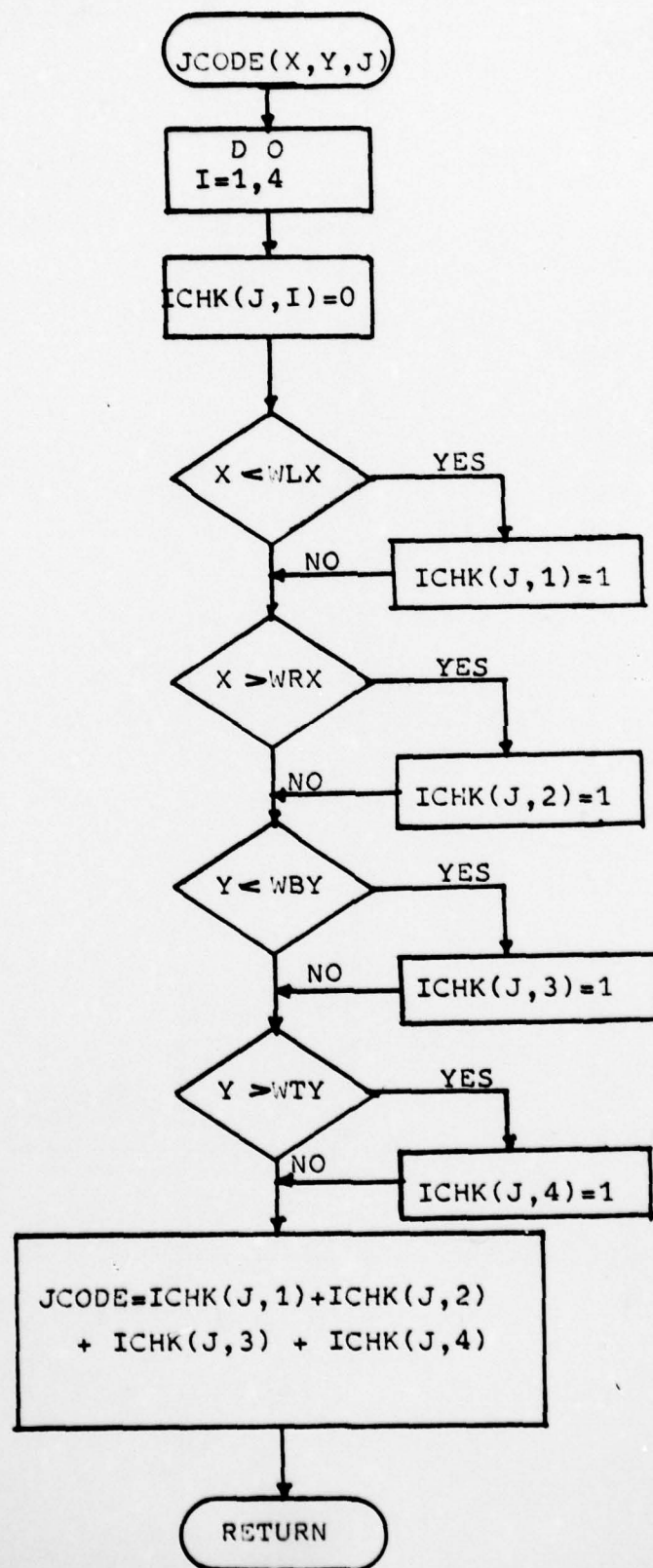
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      JCODE: TO DETERMINE IF AN END POINT OF AN EDGE IS VISIBLE
C      IN THE CURRENT WINDOW AND IF NOT COMPUTES THE POSITION OF THE
C      END POINTS ON THE SCREEN.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION JCODE(X,Y,J)
      COMMON /RG/ DELTA1,RLX,RRX,RLY,RRY
      COMMON /RI/ ICHK(2,4)
      INTEGER DELTA1
      DO 270 I=1,4
         ICHK(J,I)=0
270  CONTINUE
      IF(X.LT.RLX) ICHK(J,1)=1
      IF(X.GT.RRX) ICHK(J,2)=1
      IF(Y.LT.RLY) ICHK(J,3)=1
      IF(Y.GT.RRY) ICHK(J,4)=1
      JCODE=ICHK(J,1)+ICHK(J,2)+ICHK(J,3)+ICHK(J,4)
      RETURN
      END

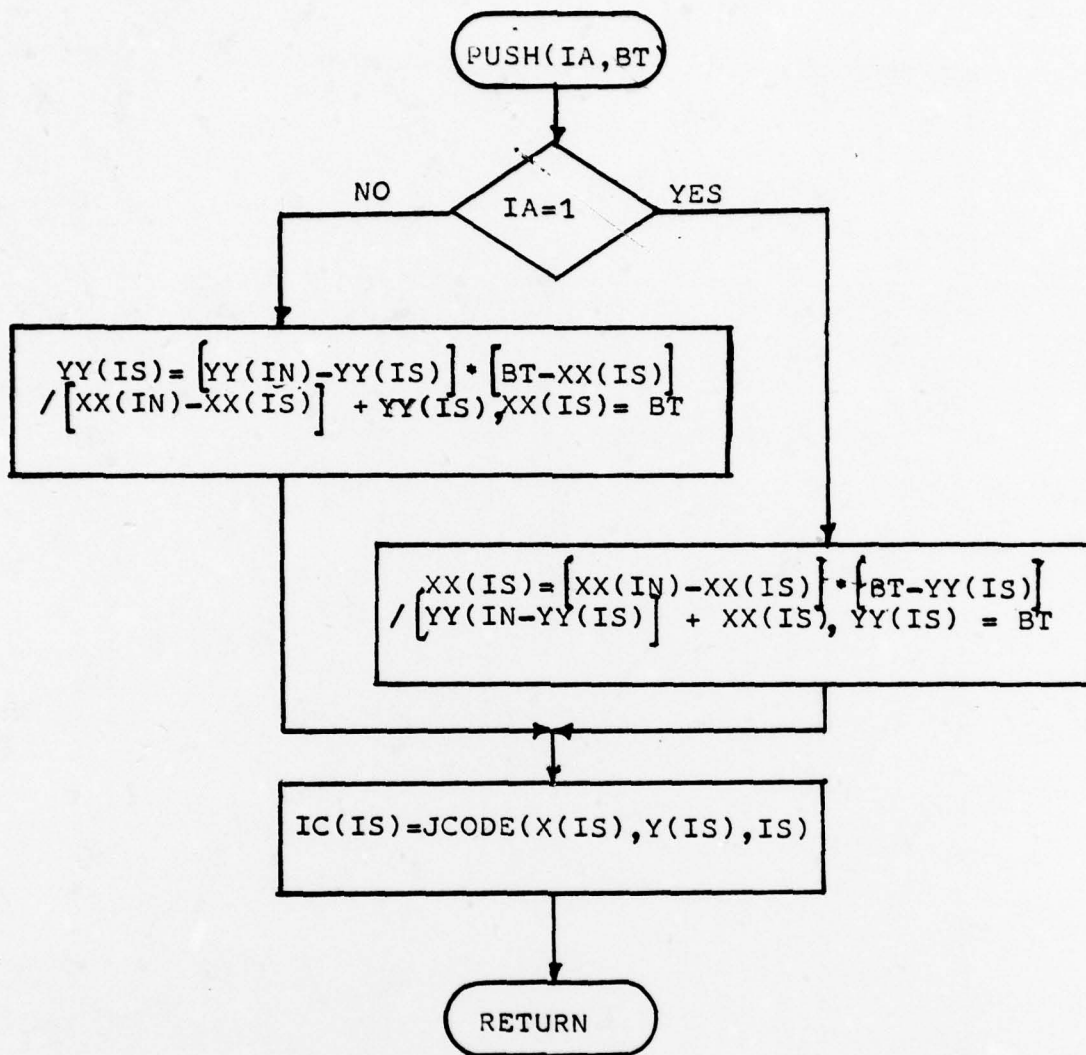
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      PUSH: USED BY CLIP2 TO DETERMINE THE VISIBLE PORTION OF THIS
C      EDGE IN THE CURRENT DISPLAY WINDOW.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE PUSH(IA,HT)
COMMON /RH/ XX(2),YY(2)
COMMON /RJ/ IC(2),IS,IN
IF(IA.EQ.1) GO TO 275
YY(IS)=(YY(IN)-YY(IS))*(HT-XX(IS))/(XX(IN)-XX(IS))+YY(IS)
XX(IS)=HT
GO TO 276
275 XX(IS)=(XX(IN)-XX(IS))*(HT-YY(IS))/(YY(IN)-YY(IS))+XX(IS)
    YY(IS)=HT
276 IC(IS)=JCODE(XX(IS),YY(IS),IS)
    RETURN
    END

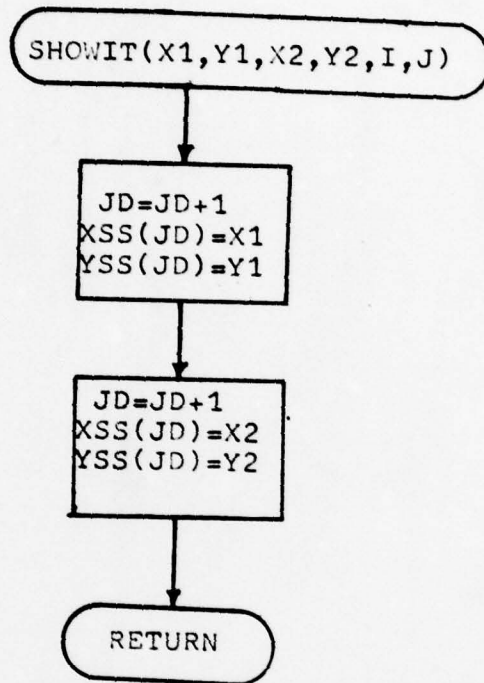
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   SHOWIT: STORES THE IMAGE IN THE APPROPRIATE ARRAYS FOR DISPLAY
C   BY SUBROUTINE DISPLY2.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE SHOWIT(X1,Y1,X2,Y2,I,J)
COMMON /DA/ XSS(2000),YSS(2000)
COMMON /DH/ JD
JD=JD+1
XSS(JD)=X1
YSS(JD)=Y1
JD=JD+1
XSS(JD)=X2
YSS(JD)=Y2
RETURN
END

```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   DISPL2 : ACTUALLY DRAWS THE IMAGE WITH THE HIDDEN LINES REMOVED
C           ON THE SELECTED DEVICE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE DISPL2(ISHAD)
COMMON /DA/ XSS(2000),YSS(2000)
COMMON /DB/ JD
INTEGER VECTOR,COLOR
DATA EPS,ACC/0.0001,200.0/
I=COLOR(ISHAD)
DO 400 I=1,JD,2
  I1=I+1
  X1=XSS(I)
  X2=XSS(I1)
  Y1=YSS(I)/2.0
  Y2=YSS(I1)/2.0
  KI=VECTOR(X1,Y1,X2,Y2)
  IF(KI.LT.0) WRITE(6,3)
400 CONTINUE
RETURN
3  FORMAT('THE FUNCTION VECTOR FAILED')
END

```

AD-A081 037

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/G 9/2

A PORTABLE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE PACKAGE--ETC(U)

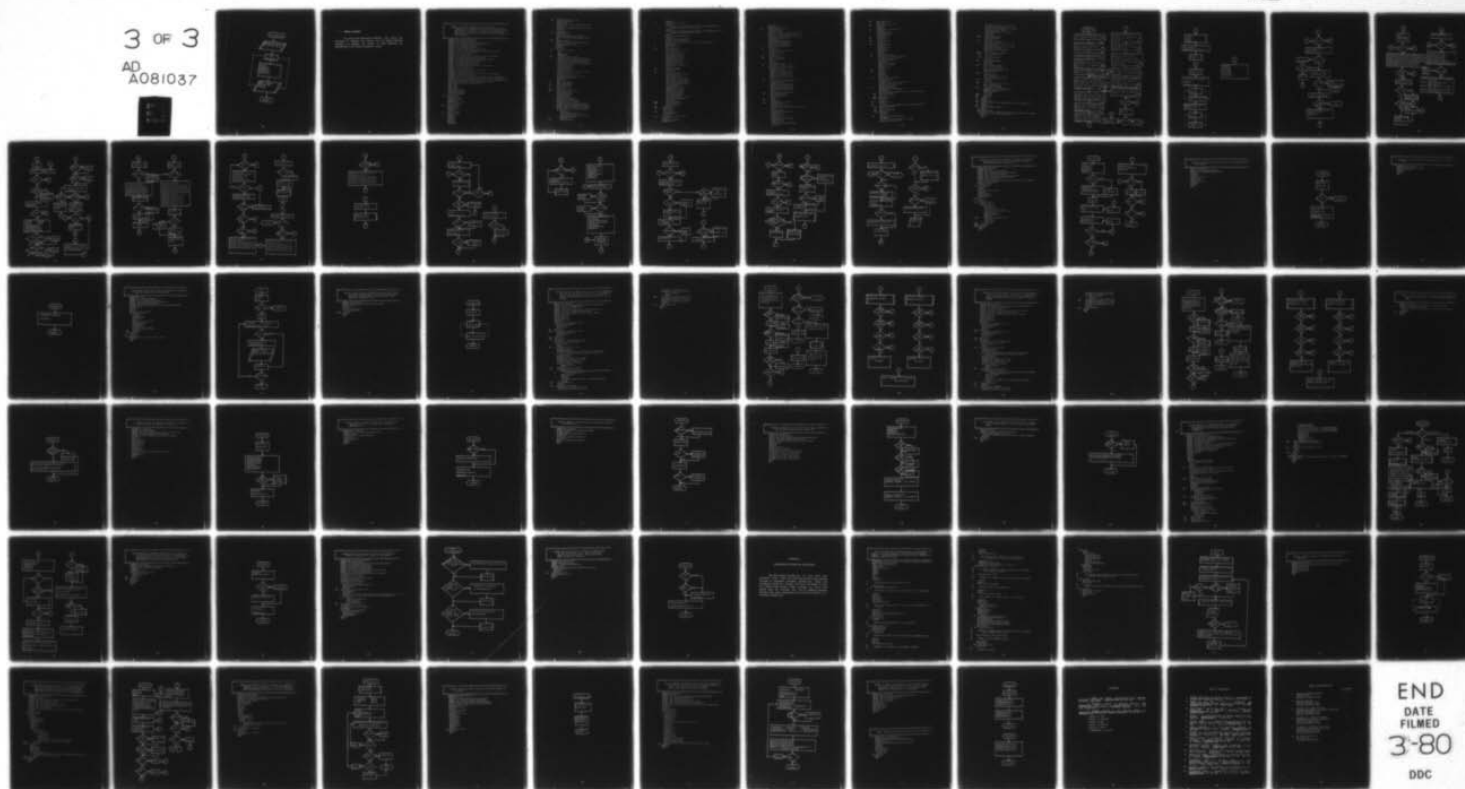
SEP 78 H J ROOD

UNCLASSIFIED

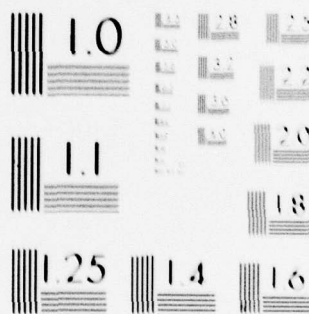
NL

3 OF 3

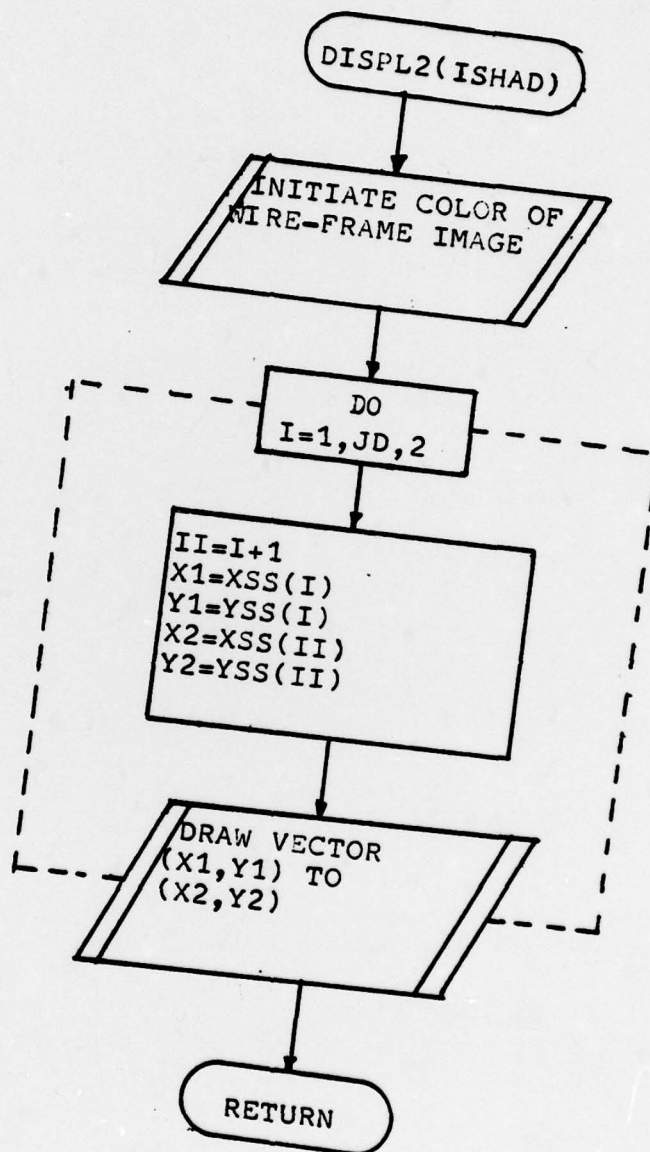
AD  
A081037



END  
DATE  
FILMED  
3-80  
DDC



MILLIMETER RESOLUTION TEST CHART  
 NATIONAL BUREAU OF STANDARDS-1963-A



### 3. Hidden Surfaces

By calling the subroutine SURFACE, this group was utilized to display the object with its hidden surfaces removed, or to display the hidden or back surfaces, as determined by the calling parameter ILOOK.

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      SURFAC: IS A MASTER SUBROUTINE WHICH CALLS THE
C      SUBROUTINES TO TRANSFORM OBJECT COORDINATES TO SCREEN
C      COORDINATES AND DISPLAYS THE OBJECT WITH SOLID SURFACES.
C      ADDITIONALLY, SURFACE CAN DISPLAY THE SURFACES WHICH
C      ARE CLOSEST TO THE VIEWER OR IT CAN DISPLAY AN OBJECT'S
C      BACK SURFACES.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE SURFAC(ISR,ILOOK)
COMMON /AAD/EDGE(2,200),EDGEN
COMMON /AAB/XS(120),YS(120),ZS(120),POINTM
COMMON /BAZENILST(200),P(2,200),EDGLST
COMMON /IAZ /IACTIVE(60),IFRELS
COMMON /IBZ/ILEFT(60),IRGHT(60)
COMMON /IC/IXLEFT(60),IXRGHT(60),SEGEST
COMMON /IDZ/XLEFT(60),XRIGHT(60),ZLEFT(60),ZRIGHT(60)
COMMON /IEZ/XSENF, XSPNRG, XRESL
COMMON /IF/IBXCNT,IBXTYP
COMMON /IGZ /XLEFT, XRIGHT, XZLEFT, XZRIGHT, BZMIN, BZMAX
COMMON /IHZ/SXLEFT, SXRGHT, SZLEFT, SZRGHT
COMMON /II/IBSEG1, IBSEG2, DIV, SDIV, INFULL, ISFULL
COMMON /ITJ/SEGSAM, IMPLST, IMPLS2, PREVIS
COMMON /IXZ/POLSEG(60), POLGON(60)
COMMON /TL/DXLEFT(60), DXRIGHT(60)
COMMON /INZ/SAMFRE, SAMLST, SAMLNK(120), SAMX(120)
COMMON /TZ/INPLET, XLSISM
COMMON /TOZ/SEGENT, LSTSEG, LY
COMMON /ABZ/POLYGN(60,11), POLGN, SHAD(60)
COMMON /JJZ/VSA, VSY, VCX, VCY
DIMENSION IYENTH(240), OZLEFT(60), OZRIGHT(60), CHANGE(60)
DIMENSION SEGLST(60), ISR(2)
INTEGER POLYGN, POLGN, SHAD, SEGLST, POLSEG, CHANGE, CHG, SEG, SEG1
INTEGER EDGE, EDGEN, ENILST, SEGEST, SAMFRE, PIR, P, PI, POLCHG, POLGON
INTEGER POINTM, EDGLST, SEGSAM, SEGENT, SEGACT, SEGLO, SEGOUT, SAMLNK
INTEGER SAMFST, SAMLST, SAMPLE, CURSEG, PREVIS, SAMX
DATA MAXSEG, APS/60, 0.00017, SAMFST/07
CALL KDYCLP
CALL CLIP
CALL SCRN
CALL INIT2(ISR)
IYRES=2.0*VSY+APS
CALL SHOWIN(IYRES)
DO 101 I=1, IYRES
101 IYENTH(I)=0
XRESL=2.0*VSX
IXRES=XRESL+APS
MM=MAXSEG-1
IACTIVE(MAXSEG)=0
DO 53 I=1, MM
53 IACTIVE(I)=1+1
MM=MAXSEG-2
SAMLNK(MM)=0
MM=MM-1
DO 54 I=1, MM
54 SAMLNK(I)=1+1
IFRELS=1
SAMFRE=1
IMPLST=0
IMPLS2=0
SEGEST=0
PIR=EDGLST

```

```

56  IF (PTR.EQ.0) GO TO 55
    NEXT=ENTLST(PTR)
    I=P(1,PTR)
    IF (I.NE.0.AND.SHAD(I).NE.0) GO TO 57
    I=P(2,PTR)
    IF (I.NE.0.AND.SHAD(I).NE.0) GO TO 57
    PTR=NEXT
    GO TO 56
57  J=EDGE(1,PTR)
    K=EDGE(2,PTR)
    IF (YS(J).LE.YS(K)) GO TO 58
    CALL ISMAP(EDGE(1,PTR),EDGE(2,PTR))
    J=K
58  I=YS(J)+0.00000
    IF (I.LI.1.0H.I.GT.IYRES) GO TO 100
    ENTLST(PTR)=IYENTR(1)
    IYENTR(1)=PTR
    PTR=NEXT
    GO TO 56
55  DO 59 I=1,IYRES
    Y=IY
    POLCHG=-1
    SEG=SEG1ST
64  IF (SEG.EQ.0) GO TO 60
    XLEFT(SEG)=XLEFT(SEG)+DXLEFT(SEG)
    YRIGHT(SEG)=YRIGHT(SEG)+DYRIGHT(SEG)
    ZLEFT(SEG)=ZLEFT(SEG)+DZLEFT(SEG)
    ZRIGHT(SEG)=ZRIGHT(SEG)+DZRIGHT(SEG)
    IY1=IYLEFT(SEG)+1
    IY2=IYRIGHT(SEG)+1
    IYLEFT(SEG)=IY1
    IYRIGHT(SEG)=IY2
    IF (IY1.NE.0.AND.IY2.NE.0) GO TO 61
    PTR=POLGON(SEG)
    IF (PTR.EQ.0) GO TO 62
    IF (CHANGE(PTR).NE.0) GO TO 61
    CHANGE(PTR)=POLCHG
    POLCHG=PTR
    GO TO 61
62  CALL XMASHT(SEG)
    CALL XEIRUK(SEG)
61  SEG=IXSNGT(SEG)
    GO TO 64
60  PTR=IYENTR(IY)
65  IF (PTR.EQ.0) GO TO 66
    IVV1=EDGE(1,PTR)
    IVV2=EDGE(2,PTR)
    IYFRST=YS(IVV1)
    IYLAST=YS(IVV2)
    IDELY=IYFRST-IYLAST
    REALDY=YS(IVV2)-YS(IVV1)
    IF (IDELY.GE.0) GO TO 67
    XSLOPE=(XS(IVV2)-XS(IVV1))/REALDY
    XFIRST=XS(IVV1)+XSLOPE*(Y-YS(IVV1))
    ZSLOPE=(ZS(IVV2)-ZS(IVV1))/REALDY
    ZFIRST=ZS(IVV1)+ZSLOPE*(Y-YS(IVV1))
    DO 68 I=1,2
    PI=P(I,PTR)
    IF (PI.EQ.0) GO TO 68
    IF (CHANGE(PI).NE.0) GO TO 69
    CHANGE(PI)=POLCHG
    POLCHG=PI
69  SEG=SEGLST(PI)

```

```

71      PREVIS=0
        IF (SEG.EQ.0) GO TO 70
        IIE1=0
        IF ((XFIRST.LT.XLEFT(SEG)).OR.((XFIRST.EQ.XLEFT(SEG)).AND.
8          (XSLOPE.LT.0XLEFT(SEG)))) IIE1=8
        IIE2=0
        IF ((XFIRST.LT.XRIGHT(SEG)).OR.((XFIRST.EQ.XRIGHT(SEG)).AND.
8          (XSLOPE.LT.0XRIGHT(SEG)))) IIE2=4
        IY1=0
        IF (IYLEFT(SEG).LT.0) IY1=2
        IY2=0
        IF (IYRIGHT(SEG).LT.0) IY2=1
        II=IIE1+IIE2+IY1+IY2+1
        GO TO (3,5,5,1,3,70,5,72,5,5,70,72,3,70,70,70),II
3        PREVIS=SEG
        SEG=POLSEG(SEG)
        GO TO 71
70      SEG1=IGIBLK(IY)
        IF (SEG1.EQ.0) GO TO 160
        POLGON(SEG1)=PI
        XLEFT(SEG1)=XFIRST
        DXLEFT(SEG1)=XSLOPE
        ZLEFT(SEG1)=ZFIRST
        DZLEFT(SEG1)=ZSLOPE
        IYLEFT(SEG1)=IYELY
        CALL PUTXST(SEG1)
        IF (PREVIS.EQ.0) GO TO 73
        POLSEG(PREVIS)=SEG1
        GO TO 74
73      SEGLST(PI)=SEG1
74      POLSEG(SEG1)=SEG
        GO TO 68
72      SEG1=IGIBLK(IR)
        IF (SEG1.EQ.0) GO TO 160
        POLGON(SEG1)=PI
        XLEFT(SEG1)=XLEFT*(SEG)
        DXLEFT(SEG1)=DXLEFT*(SEG)
        ZLEFT(SEG1)=ZLEFT*(SEG)
        DZLEFT(SEG1)=DZLEFT*(SEG)
        IYLEFT(SEG1)=IYLEFT*(SEG)
        IYLEFT(SEG1)=0
        XRIGHT(SEG1)=XFIRST
        DXRIGHT(SEG1)=XSLOPE
        ZRIGHT(SEG1)=ZFIRST
        DZRIGHT(SEG1)=ZSLOPE
        IYRIGHT(SEG1)=IYELY
        CALL PUTXST(SEG1)
        IF (PREVIS.EQ.0) GO TO 608
        POLSEG(PREVIS)=SEG1
        GO TO 609
608      SEGLST(PI)=SEG1
609      POLSEG(SEG1)=SEG
        PREVIS=SEG1
68      CONTINUE
67      PTR=ENTLST(PTR)
        GO TO 65
93      PREVIS=SEG
        SEG=POLSEG(SEG)
        GO TO 76
66      IF (POLCHG.EQ.-1) GO TO 75
        PI=POLCHG
        POLCHG=CHANGE(PI)
        CHANGE(PI)=0

```

```

PREVIS=0
SEG=SEGLST(PT)
76 IF (SEG.EQ.0) GO TO 66
   IY1=IYLEFT(SEG)
   IY2=IYRIGHT(SEG)
   IF (IY1.LI.0.AND.IY2.LI.0) GO TO 93
   IF (IY1.EQ.0.AND.IY2.EQ.0) GO TO 77
   IF (IY1.NE.0.OR.IY2.GE.0) GO TO 78
   IYLEFT(SEG)=IYRIGHT(SEG)
   IYRIGHT(SEG)=0
   XLEFT(SEG)=XRIGHT(SEG)
   DXLEFT(SEG)=DXRIGHT(SEG)
   ZLEFT(SEG)=ZRIGHT(SEG)
   DZLEFT(SEG)=DZRIGHT(SEG)
   GO TO 76
77 I=PULSEG(SEG)
   IF (PREVIS.EQ.0) GO TO 79
   PULSEG(PREVIS)=1
   GO TO 80
79 SEGLST(PT)=1
80 CALL RMXSRT(SEG)
   CALL RETBLK(SEG)
   SEG=1
   GO TO 76
78 NEXT=PULSEG(SEG)
   IF (NEXT.EQ.0) GO TO 110
   IF (IYLEFT(NEXT).GE.0) GO TO 81
   IYRIGHT(SEG)=IYLEFT(NEXT)
   IYLEFT(NEXT)=0
   XRIGHT(SEG)=XLEFT(NEXT)
   DXRIGHT(SEG)=DXLEFT(NEXT)
   ZRIGHT(SEG)=ZLEFT(NEXT)
   DZRIGHT(SEG)=DZLEFT(NEXT)
   GO TO 76
81 IF (IYRIGHT(NEXT).GE.0) GO TO 82
   IYRIGHT(SEG)=IYRIGHT(NEXT)
   IYRIGHT(NEXT)=0
   XRIGHT(SEG)=XRIGHT(NEXT)
   DXRIGHT(SEG)=DXRIGHT(NEXT)
   ZRIGHT(SEG)=ZRIGHT(NEXT)
   DZRIGHT(SEG)=DZRIGHT(NEXT)
   GO TO 76
82 PULSEG(SEG)=PULSEG(NEXT)
   CALL RMXSRT(NEXT)
   CALL RETBLK(NEXT)
   GO TO 76
75 CHG=0
   SEG=SEGST
84 IF (SEG.EQ.0) GO TO 83
   I=IXSNGI(SEG)
   IF (I.EQ.0) GO TO 83
   IF (XLEFT(SEG).LE.XLEFT(I)) GO TO 85
   CHG=1
   K=IXSLFI(SEG)
   IF (K.NE.0) IXSNGI(K)=1
   IXSLFI(I)=K
   IXSLFI(SEG)=1
   K=IXSNGI(I)
   IF (K.NE.0) IXSLFI(K)=SEG
   IXSNGI(SEG)=K
   IXSNGI(I)=SEG
   IF (SEGST.EQ.SEG) SEGST=1
   GO TO 84

```

```

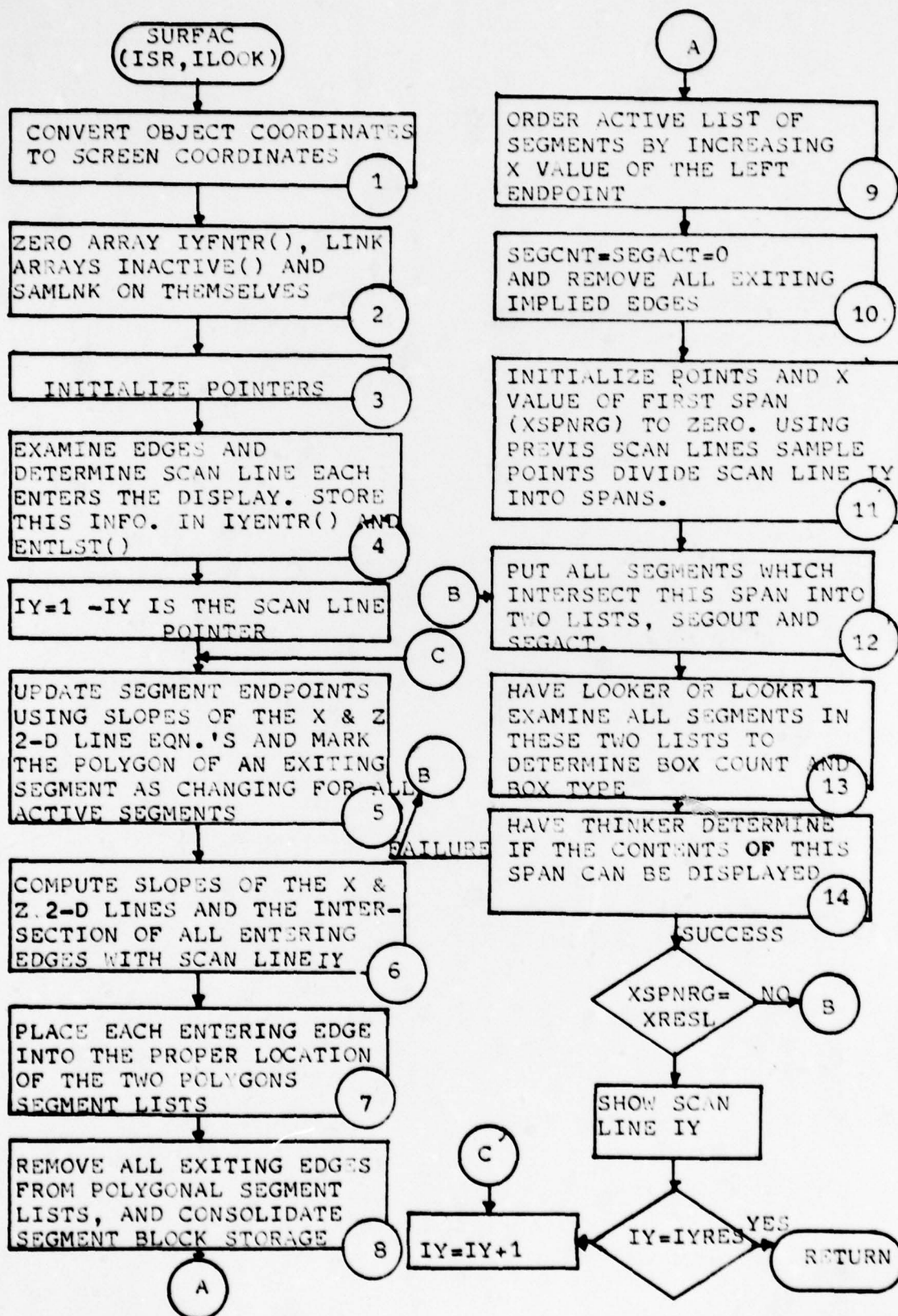
85     SEG=IXSRGT(SEG)
      GO TO 84
83     IF (CHG.EQ.1) GO TO 75
      SEGACT=0
      SEGCT=0
87     IF (IMPLST.EQ.0) GO TO 86
      J=IMPLST
      IMPLST=IXSRGT(J)
      CALL RETBLK(J)
      GO TO 87
89     XSPNRG=XRESL
      GO TO 90
88     XSPNRG=XLSTUD
      GO TO 90
86     IMPLST=IMPLS2
      IMPLS2=0
      CURSEG=SEGFST
      XSPNRG=0.0
      SAMPLE=SAMFST
      SAMLST=0
      XLSTUD=0.0
98     XSPNLF=XSPNRG+1.0
      IF (XSPNLF.LE.XLSTUD) GO TO 88
      IF (SAMPLE.EQ.0) GO TO 89
      XSPNRG=SAMX(SAMPLE)
      IX=SAMPLE
      SAMPLE=SAMPLN(IX)
      SAMLN(IX)=SAMFRE
      SAMFRE=IX
      XLSTUD=XSPNRG
90     IMPLT=0
91     IBXCT=0
      SEGOUT=0
      PREVIS=0
      SEG=SEGACT
92     IF (SEG.EQ.0) GO TO 94
      NEXT=IACTVE(SEG)
      XXX=XSPNRG+1.0
      IF (XRIGHT(SEG).GE.XXX) GO TO 95
      IF (PREVIS.EQ.0) GO TO 96
      IACTVE(PREVIS)=NEXT
97     IACTVE(SEG)=SEGOUT
      IF (SEGOUT.EQ.0) SEGLO=SEG
      SEGOUT=SEG
      IF (XRIGHT(SEG).GE.XSPNLF) GO TO (800,801),ILOOK
800    CALL LOOKER(SEG)
      GO TO 802
801    CALL LOOKRT(SEG)
802    SEG=NEXT
      GO TO 92
96     SEGACT=NEXT
      GO TO 97
95     IF (XLEFT(SEG).LE.XSPNRG) GO TO (803,804),ILOOK
803    CALL LOOKER(SEG)
      GO TO 805
804    CALL LOOKRT(SEG)
805    PREVIS=SEG
      SEG=NEXT
      GO TO 92
94     IF (CURSEG.EQ.0) GO TO 104
      SEG=CURSEG
      IF (XLEFT(SEG).GT.XSPNRG) GO TO 104
      CURSEG=IXSRGT(CURSEG)

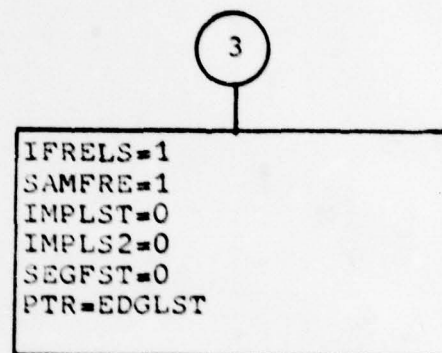
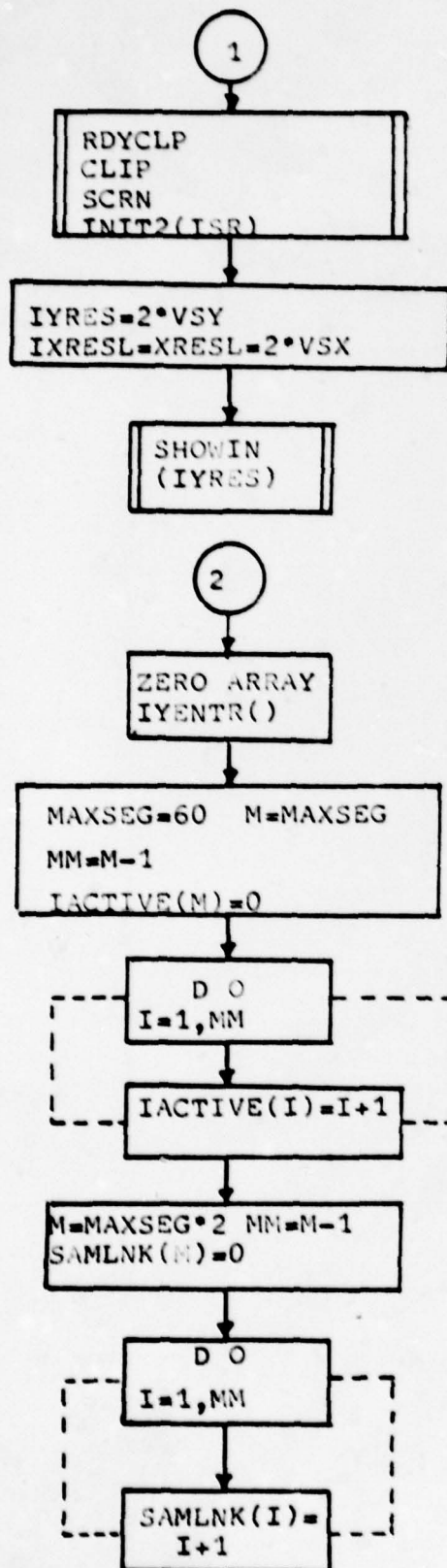
```

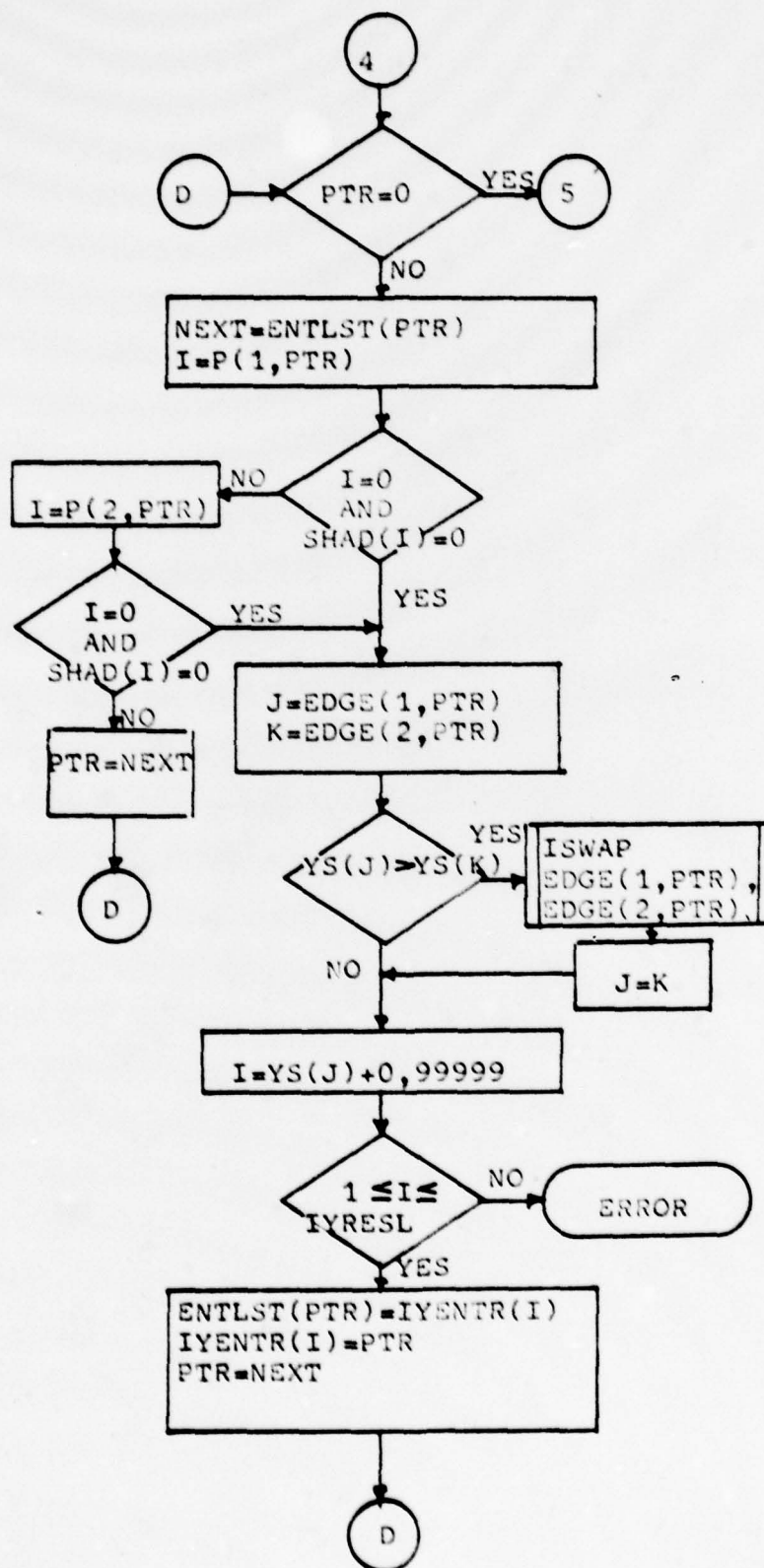
```

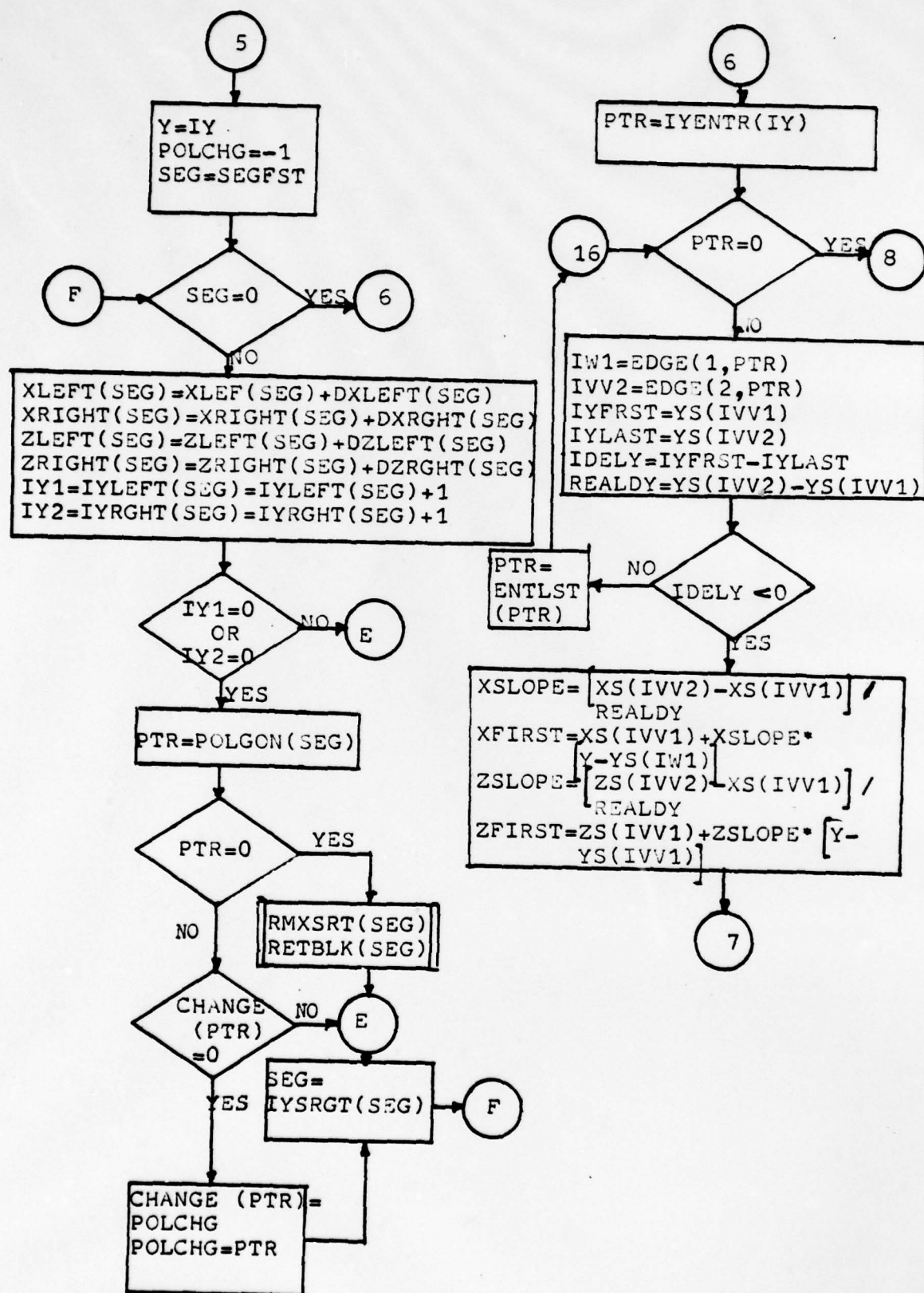
      IF (POLGON(SEG).NE.0) GO TO 99
      IF (ALEFT(SEG).LT.1.0) GO TO 600
      IF (ALEFT(SEG).GT.XRESL) GO TO 600
      IPD=PULSEG(SEG)/10000
      IF (IPD.NE.LSTSEG) GO TO 600
      INPLFT=SEG
      GO TO 94
600   CALL RMXSRT(SFG)
      CALL NEIBLK(SEG)
      GO TO 94
99    XXX=XLEFT(SEG)+1.0
      IF (XXX.GE.XRIGHT(SEG)) GO TO 94
      XSS=XSPNRG+1.0
      IF (XRIGHT(SEG).GE.XSS) GO TO 601
      IACTIVE(SEG)=SEGOUT
      IF (SEGOUT.EQ.0) SEGLO=SEG
      SEGOUT=SEG
602   GO TO (R06,R07),ILOOK
806   CALL LOOKEN(SEG)
      GO TO 94
807   CALL LOOKRT(SEG)
      GO TO 94
601   IACTIVE(SEG)=SEGACT
      SEGACT=SEG
      GO TO 602
108   CALL THINKR(1TR,SEG)
      GO TO (603,160),11R
      IF (SEGOUT.EQ.0) GO TO 604
      IACTIVE(SEGLO)=SEGACT
      SEGACT=SEGOUT
604   I=0IV
      ISSRGT=XSPNRG
      IF (I.LT.ISSRGT) GO TO 605
      I=(XSPNLF+XSPNRG)/2.0
605   XSPNRG=I
      GO TO 91
603   IF (INPLFT.EQ.0) GO TO 606
      CALL RMXSRT(INPLFT)
      CALL NEIBLK(INPLFT)
606   IF (XSPNRG.LT.XRESL) GO TO 98
      IF (SAMLST.EQ.0) GO TO 610
      SAMLNK(SAMLST)=0
      GO TO 611
610   SAMPST=0
611   CALL SHOW
59    CONTINUE
      RETURN
100   WRITE(6,121)
121   FORMAT('ERROR: THE Y COORDINATE IS OFF THE SCREEN.')
      RETURN
110   WRITE(6,111)
111   FORMAT('ERROR: NEXT=0')
      RETURN
160   WRITE(6,999)
999   FORMAT(2X,'THE NUMBER OF SEGMENTS GENERATED FOR THIS SCAN
&LINE',/2X,'THE STORAGE PROVIDED.')
      RETURN
      END

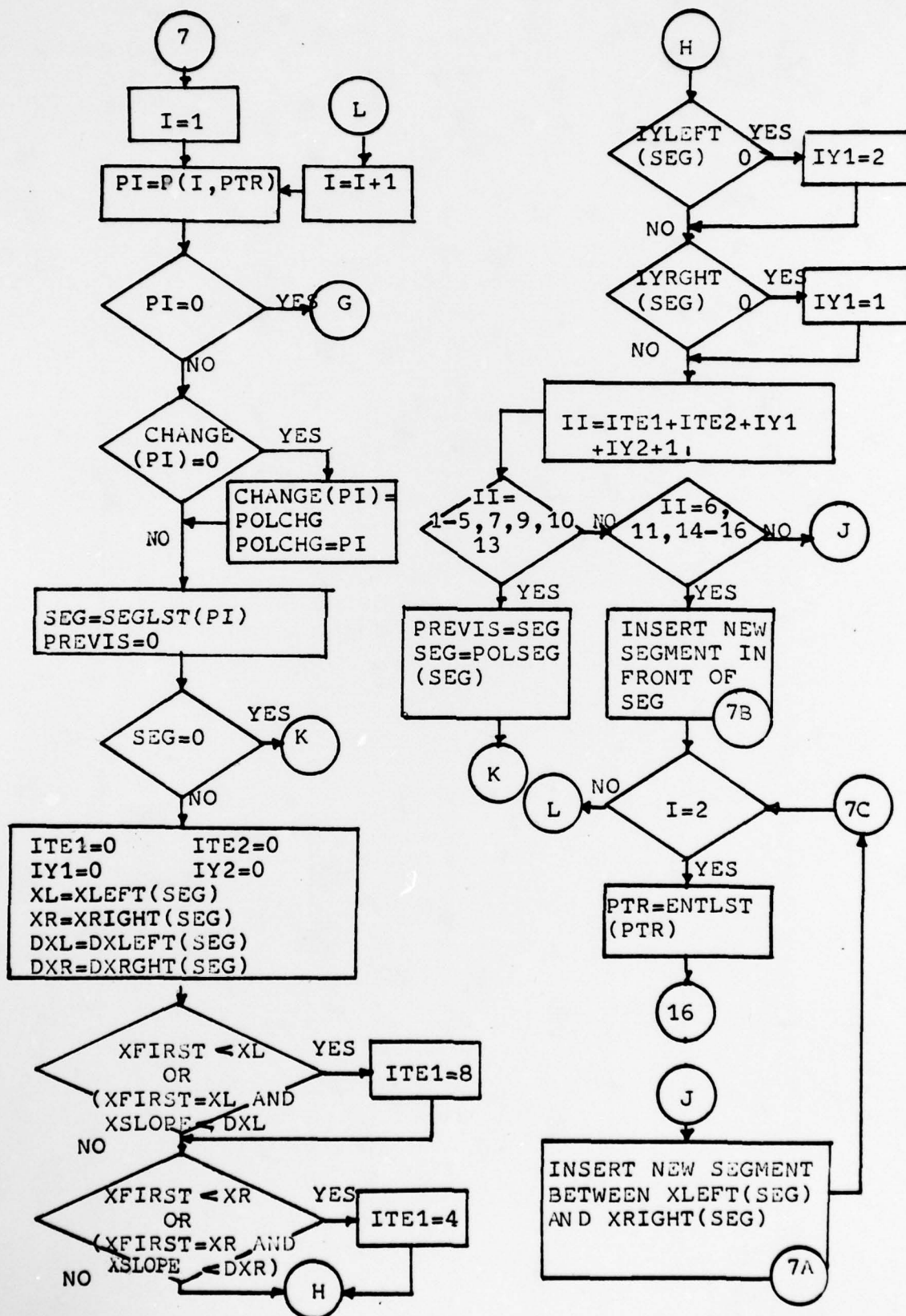
```

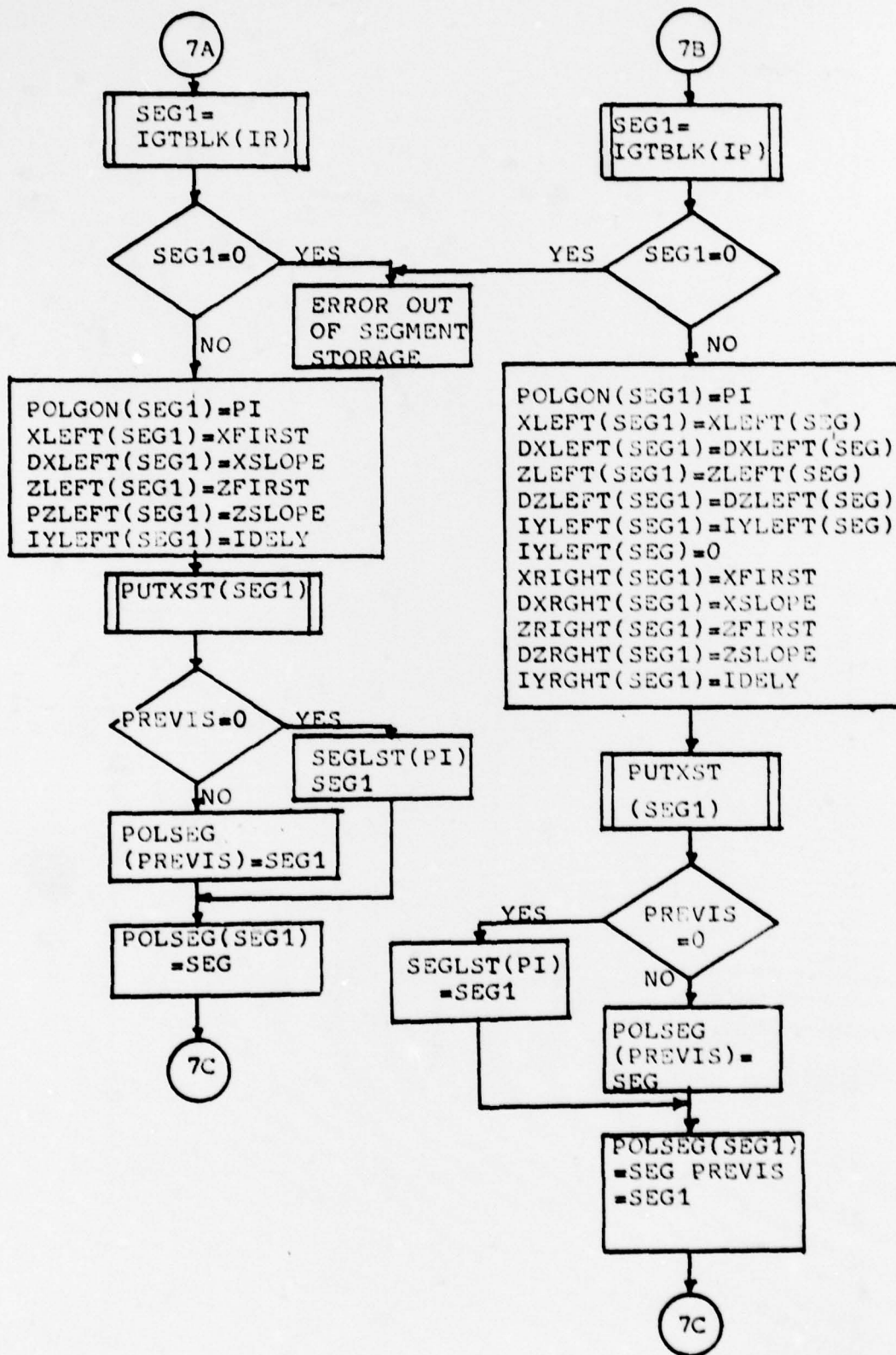


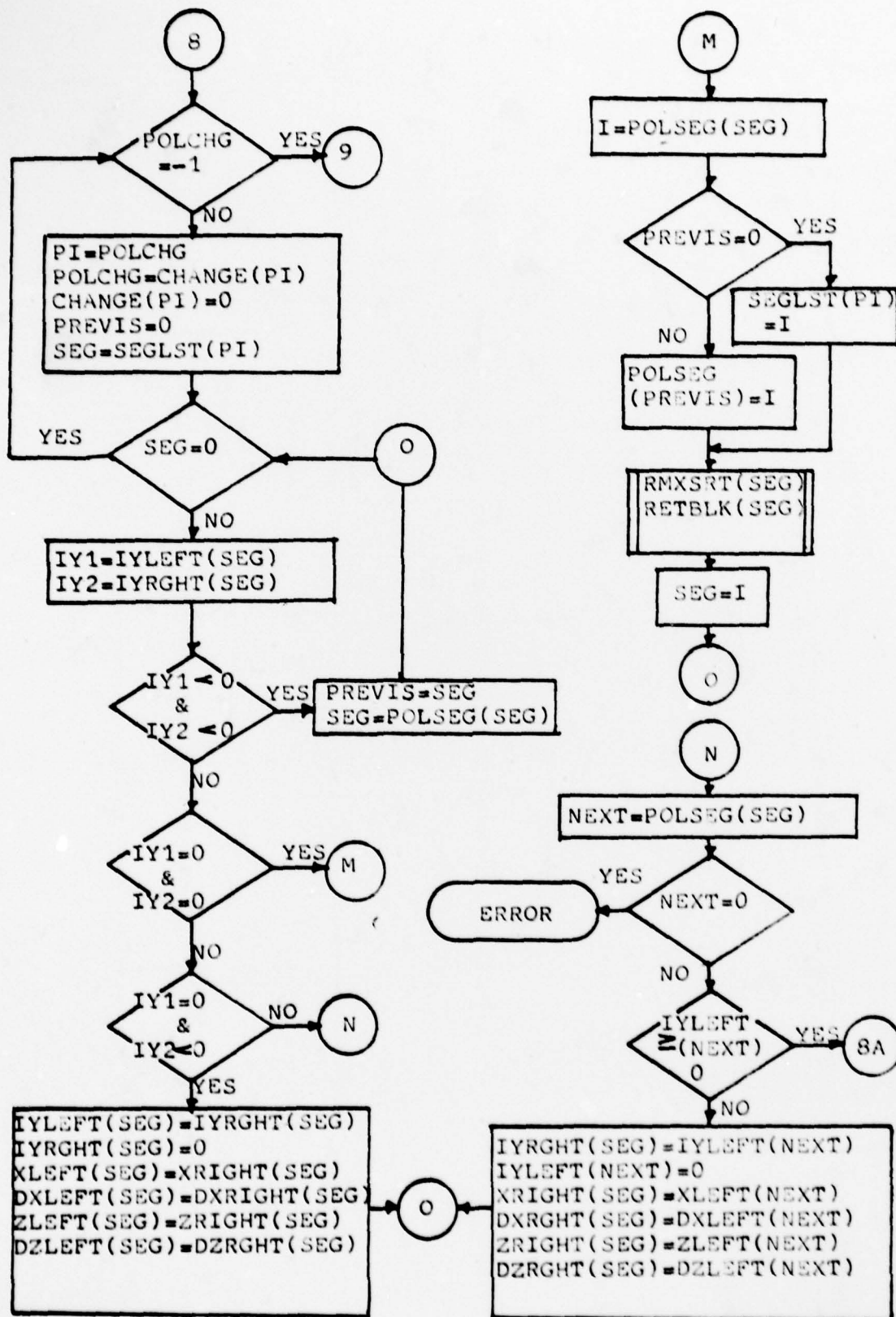


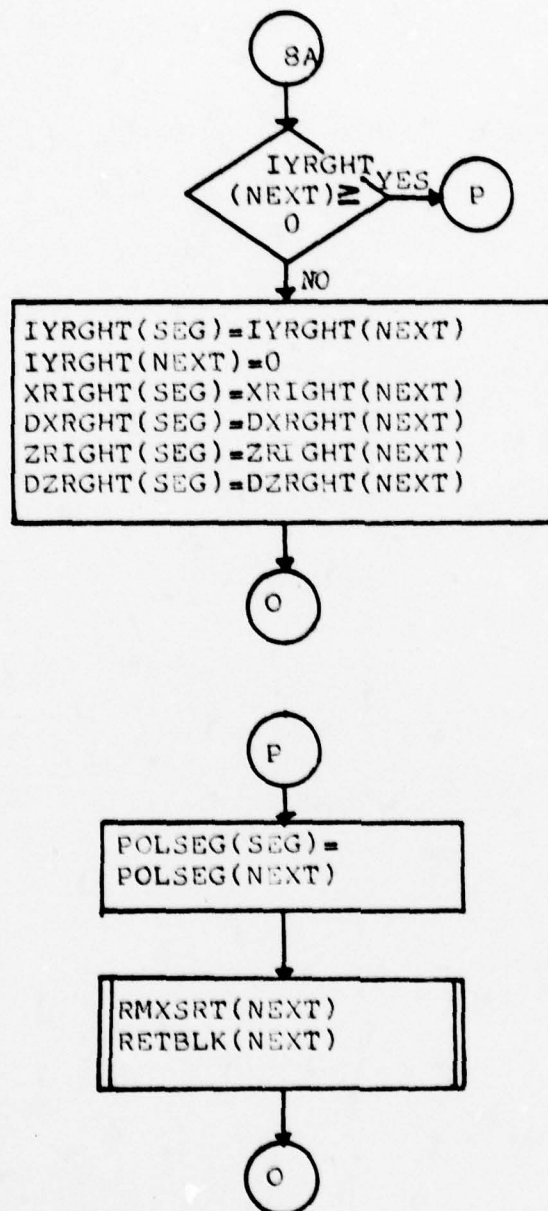


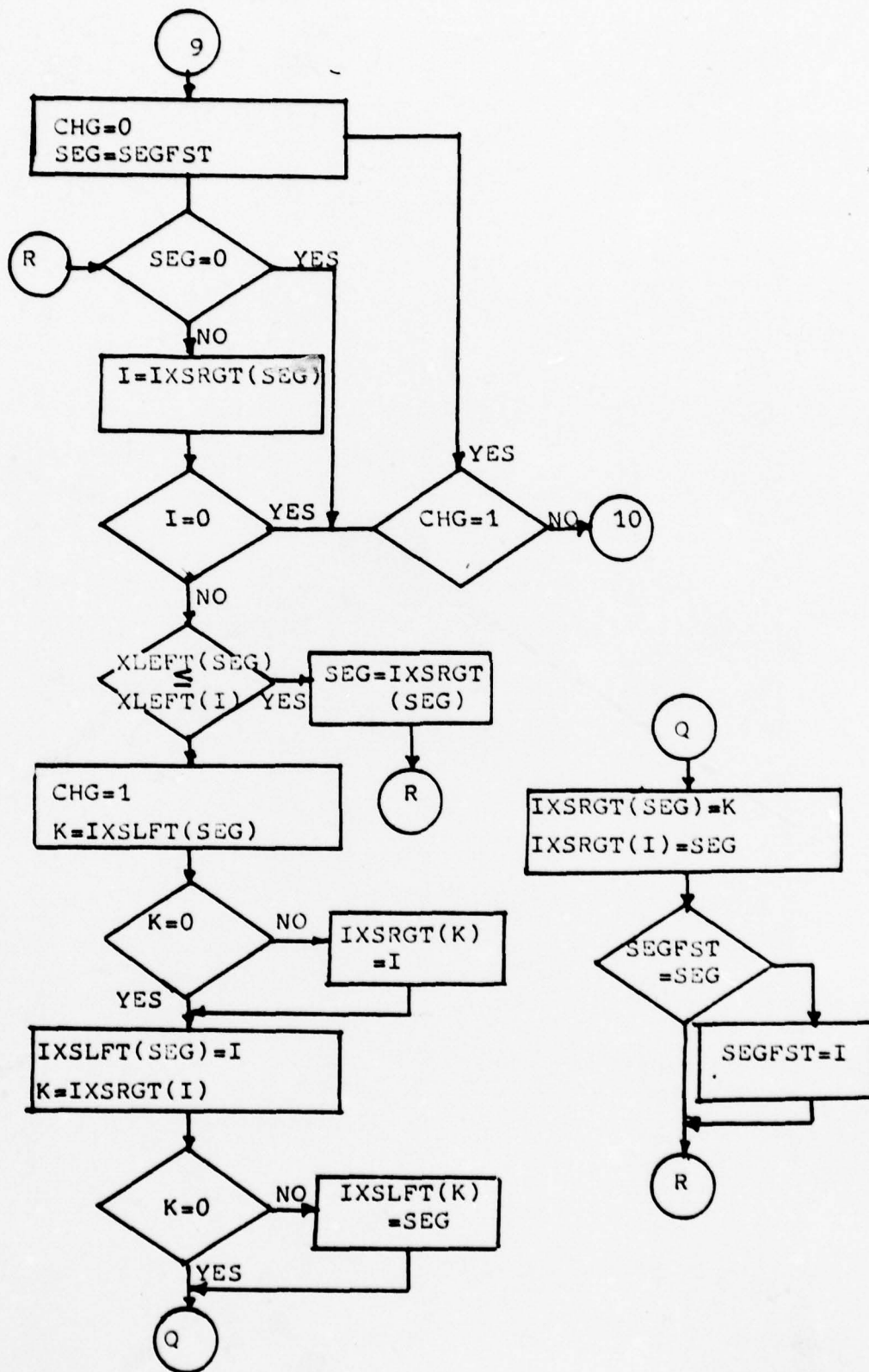


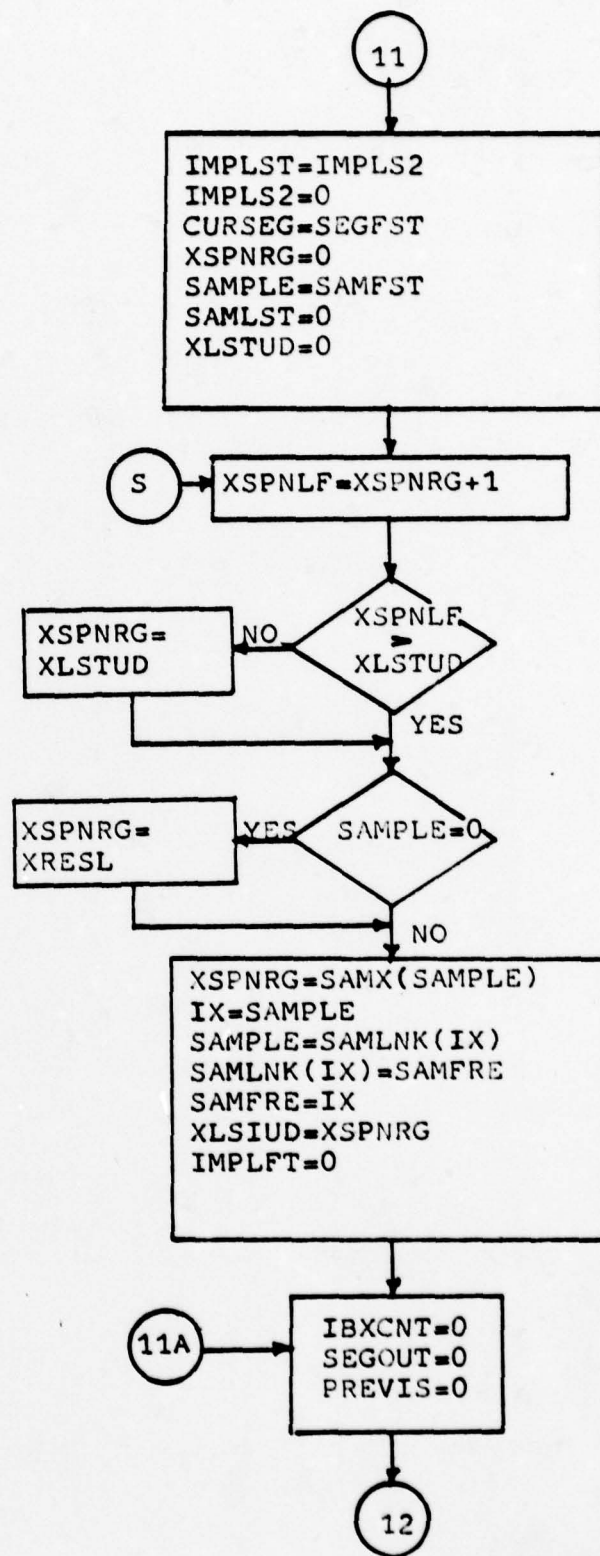
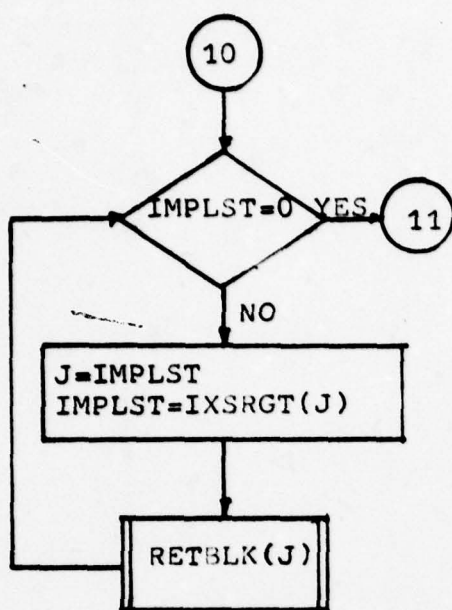


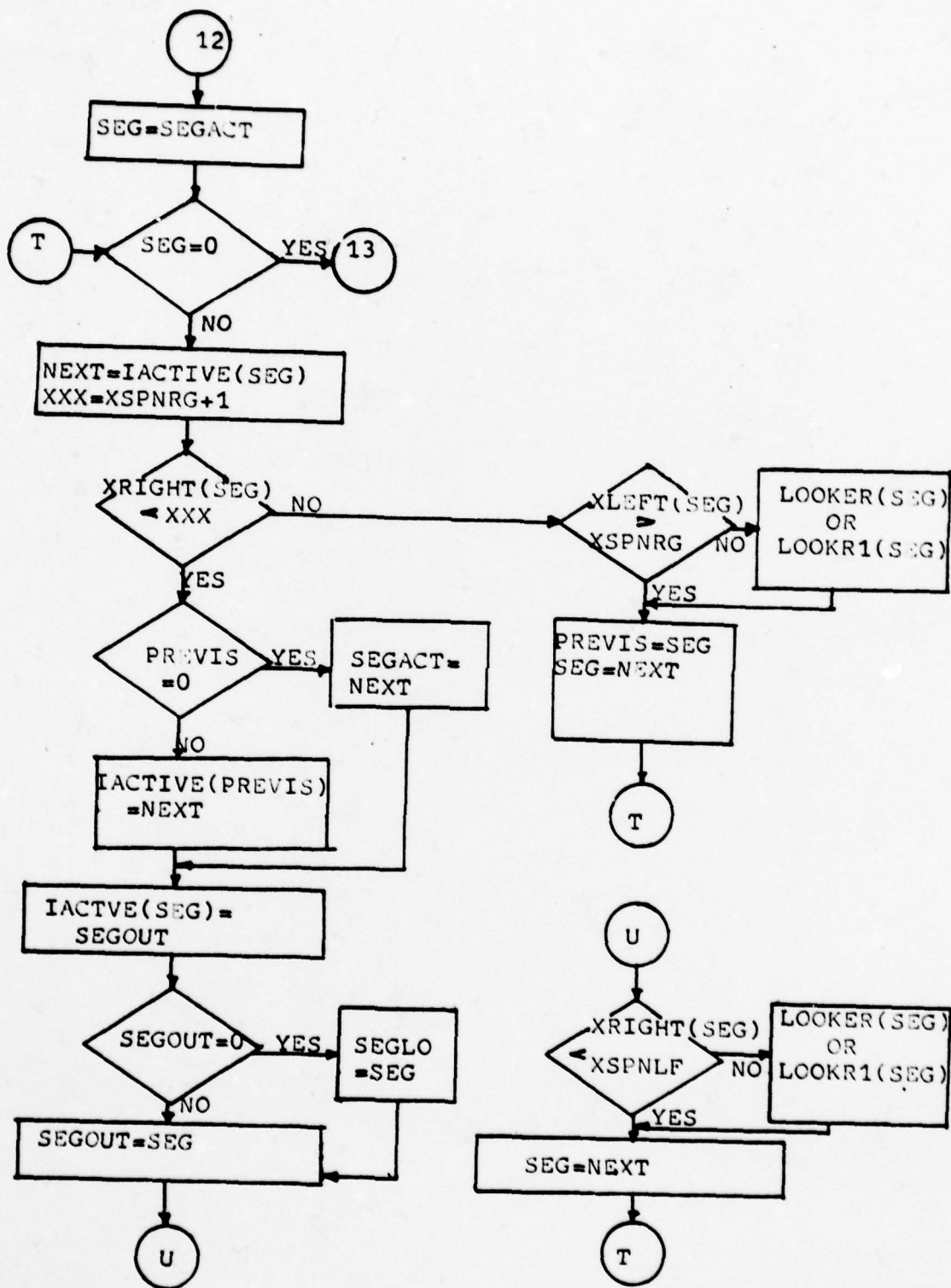


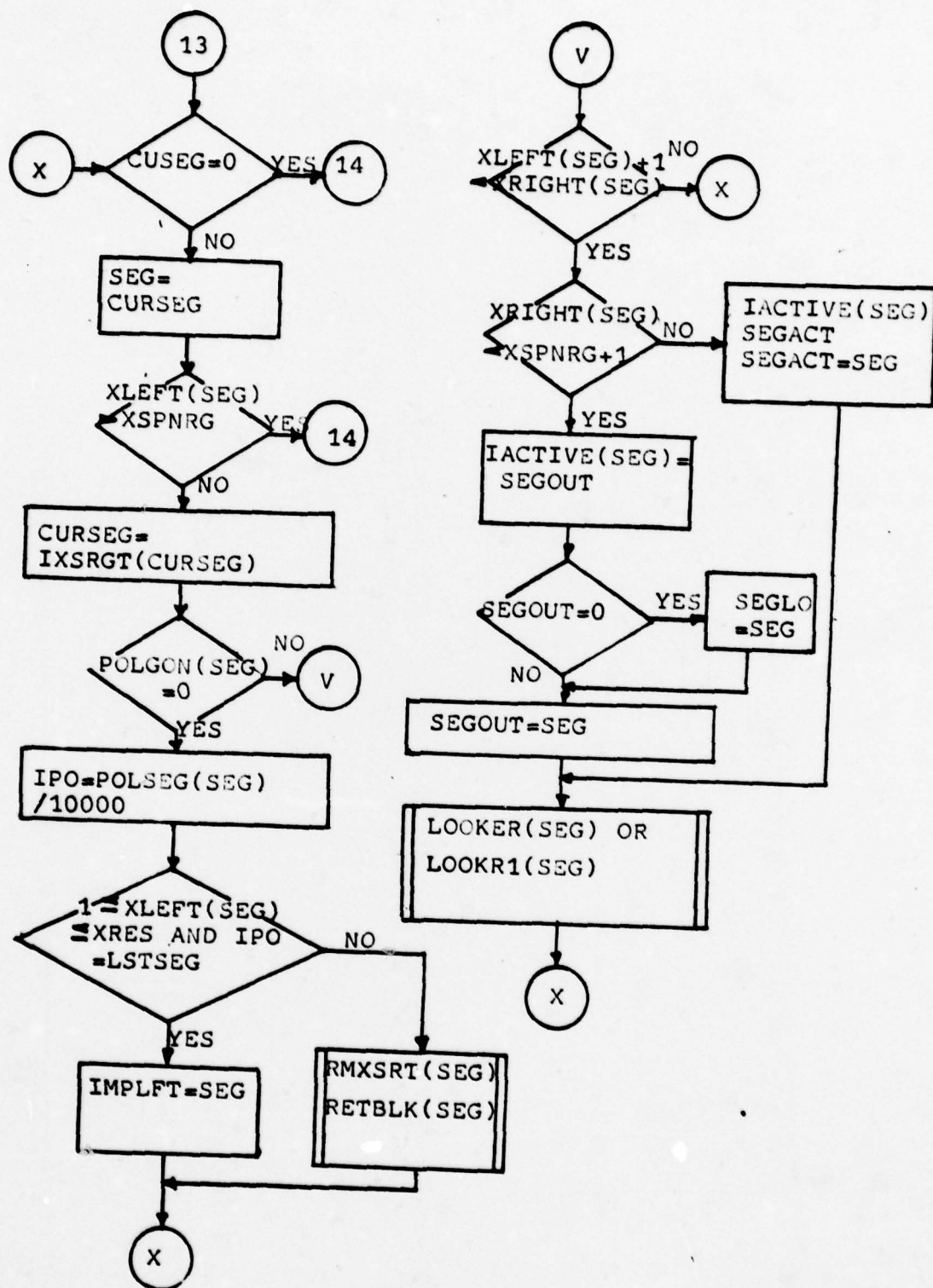


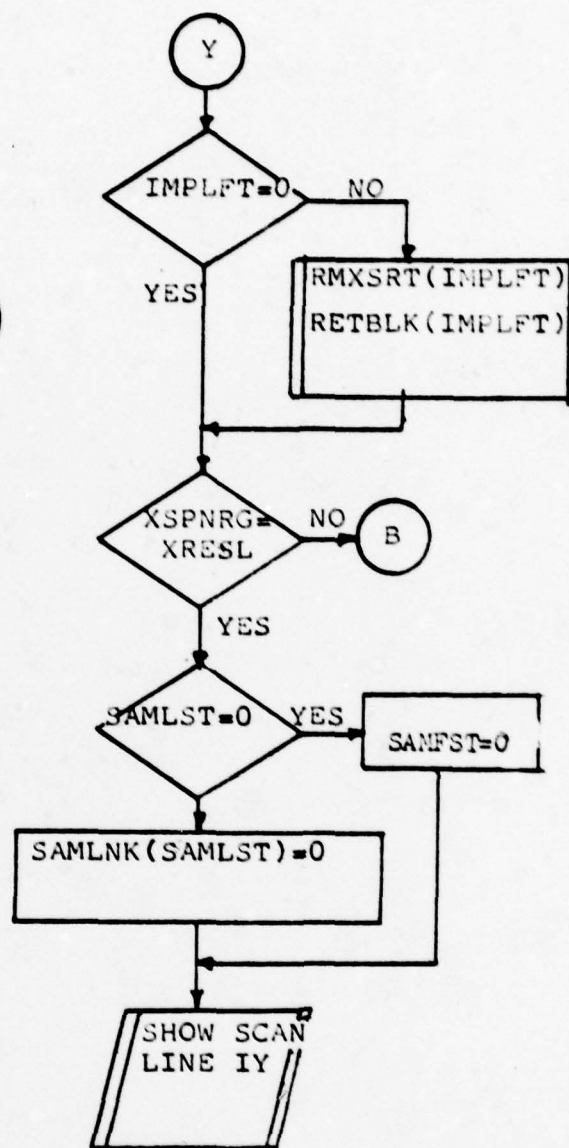
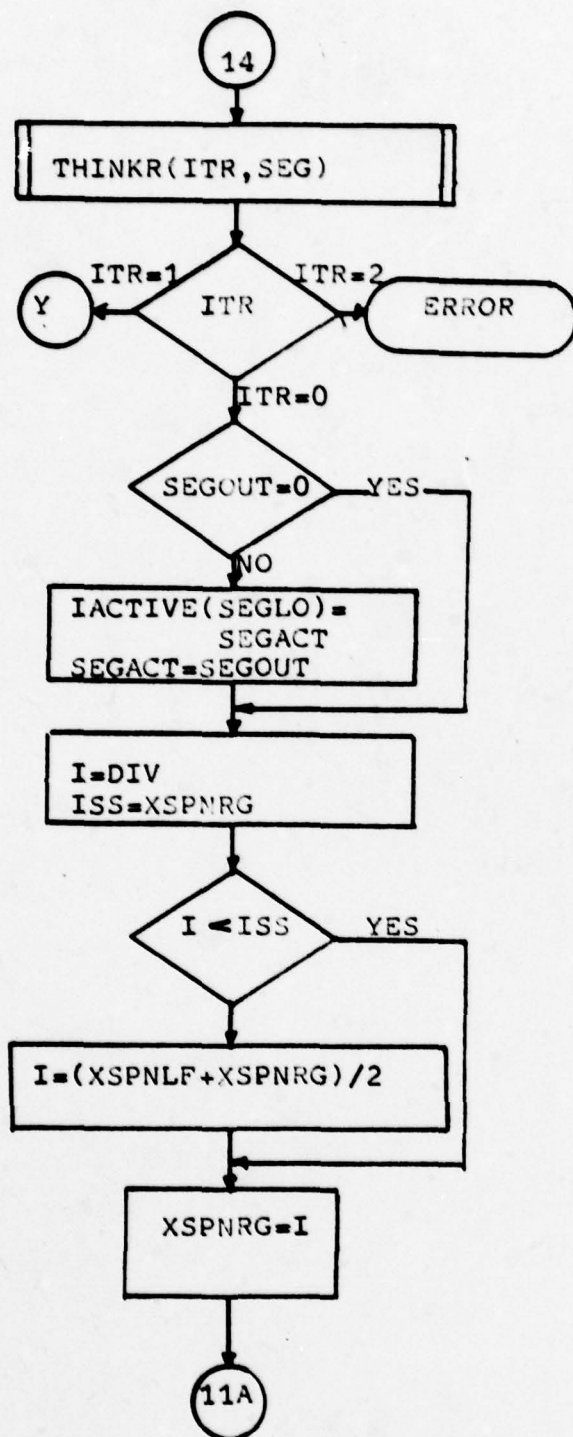








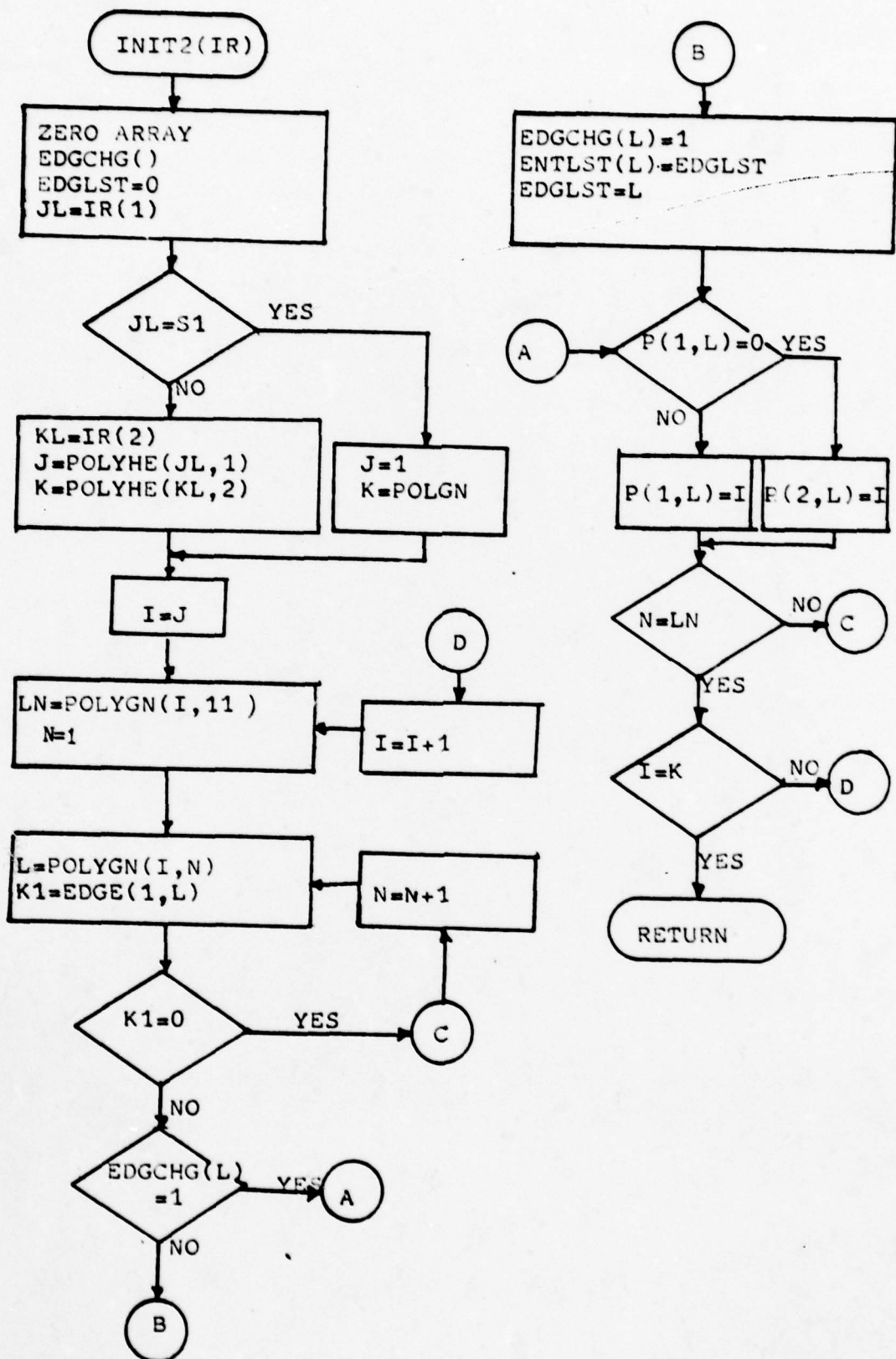




```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      INIT2: LINKS THE LIST OF EDGES IN THE ARRAY ENLST(1) AND
C      STORES THE INDEX OF THE POLYGONS WHICH HAVE EDGE I
C      AS A BOUNDARY IN THE ARRAY P(2,1).
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE INIT2(IR)
      DIMENSION IR(2)
      COMMON /AA/ POLYHE(10,2),POLYHN
      COMMON /AB/ POLYGN(60,11),POLGN,SHAD(60)
      COMMON /AAD/ EDGE(2,200),EDGEM
      COMMON /AAH/ XS(120),YS(120),ZS(120),POINTM
      COMMON /BA/ENLST(200),P(2,200),EDGLST
      COMMON /FF/EDGCHG(200)
      INTEGER EDGE,EDGEN,POLYGN,POLGN,POINTM,SHAD,ENLST,P,EDGLST,
&POLYHE,POLYHN,EDGCHG
      DO 30 I=1,EDGEM
30    EDGCHG(I)=0
      EDGLST=0
      JL=IR(1)
      IF(JL.EQ.31)GO TO 1000
      KL=IR(2)
      J=POLYHE(JL,1)
      K=POLYHE(KL,2)
      GO TO 500
1000 J=1
      K=POLGN
      DO 1010 I=J,K
      LN=POLYGN(I,11)
      DO 1020 N=1,LN
      L=POLYGN(I,N)
      K1=EDGE(I,L)
      IF(K1.EQ.0) GO TO 1020
      IF(EDGCHG(L).EQ.1) GO TO 520
      EDGCHG(L)=1
      ENLST(L)=EDGLST
      EDGLST=L
520    IF(P(1,L).EQ.0) GO TO 521
      P(2,L)=J
      GO TO 1020
521    P(1,L)=1
1020    CONTINUE
1010    CONTINUE
      RETURN
      END

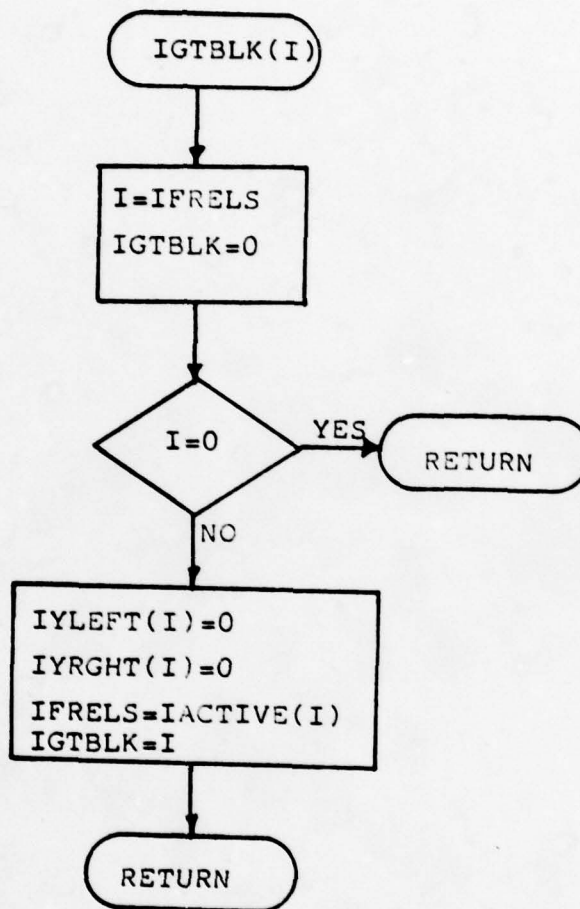
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      IGTBLK: IS USED TO OBTAIN THE INDEX OF A FREE SEGMENT BLOCK
C      OF STORAGE.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
FUNCTION IGTBLK(I)
COMMON /IA/ IACTIVE(60), IFRELS
COMMON /IB/ IYLEFT(60), IYRGHT(60)
I=IFRELS
IGTBLK=0
IF(I.EQ.0) RETURN
IYLEFT(I)=0
IYRGHT(I)=0
IFRELS=IACTIVE(I)
IGTBLK=I
RETURN
END

```



CC

C

REIBLK: RETURNS A SEGMENT'S BLOCK OF STORAGE TO THE FREE  
LIST.

C

C

CC

SUBROUTINE REIBLK(I)

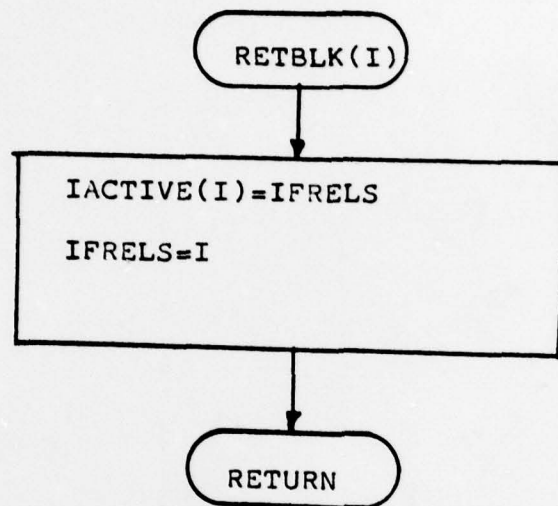
COMMON /TA/ IACTIVE(60), IFRELS

IACTIVE(I)=IFRELS

IFRELS=I

RETURN

END



CC

C  
C  
C  
C

SHOW: DISPLAYS EACH SEGMENT WITH THE APPROPRIATE POLYGONAL  
COLOR OR SHADE ON THE RAMTEK SCREEN.

CC

SUBROUTINE SHOW

COMMON /TU/ SEGNT,LSTSEG,IY

COMMON /TK/ POLSEG(60),POLGON(60)

COMMON /AB/ POLYGN(60,11),POLGN,SHAD(60)

COMMON /TP/ISPOS(60),ISSEG(60)

INTEGER SEGNT,POLSEG,POLGON,POLYGN,POLGN,SHAD,VECTOR

INTEGER COLOR

ISAMP=0

YI=IY

IF(SFGCNT.LI.1) RETURN

DO 700 I=1,SEGNT

  X1=ISAMP

  ISEG=ISSEG(I)

  IX=ISPOS(I)

  X2=IX

  IF(ISEG.EQ.0) GO TO 710

  IP=POLGON(ISEG)

  JLM=SHAD(IP)

  II=COLOR(JLM)

  KL=VECTOR(X1,Y1,X2,Y1)

  IF(KL.LT.0) GO TO 1554

710   ISAMP=IX

700 CONTINUE

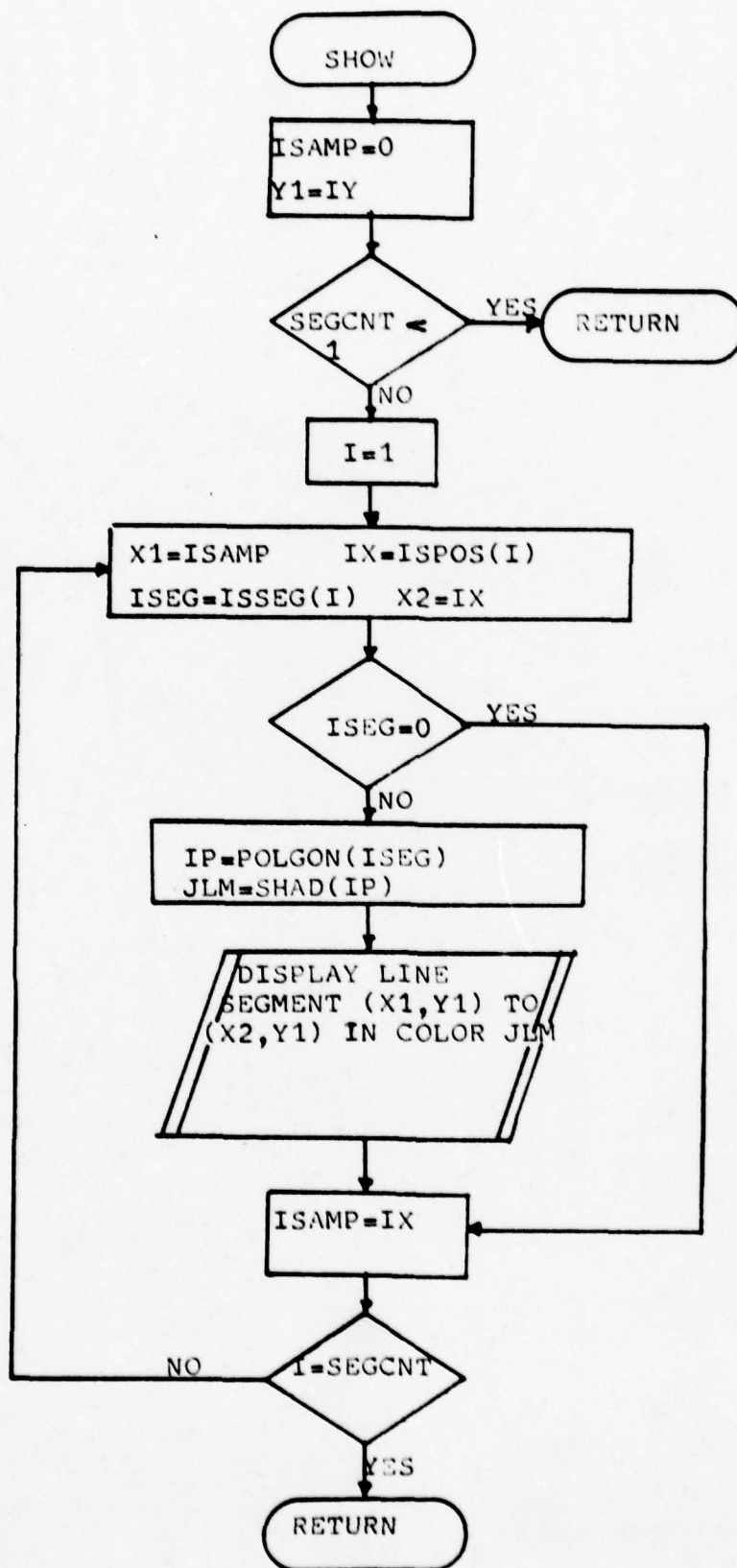
RETURN

1554 WRITE(6,1)

1 FORMAT(2X,'THE FUNCTION VECTOR FAILED')

RETURN

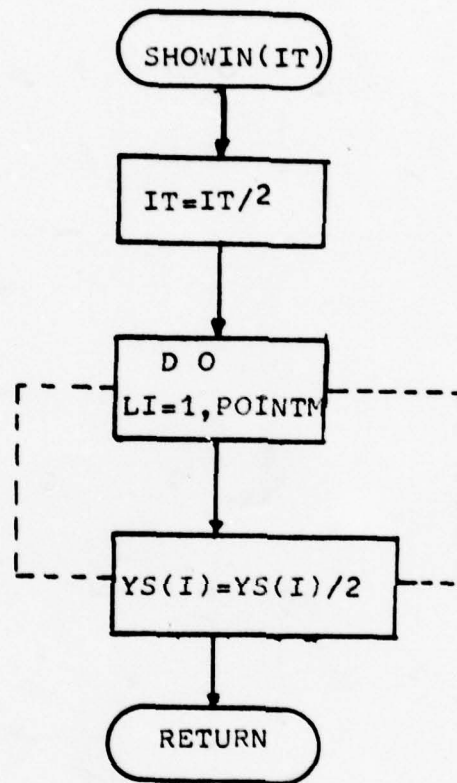
END



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      SHOWIN: DIVIDES THE YS COORDINATE VALUES BY TWO IN ORDER
C      TO DISPLAY AN UN-DISTORTED IMAGE ON THE RANIER AT THE
C      NAVAL POST GRADUATE SCHOOL WHICH HAS DOUBLE WIDE
C      HORIZONTAL RESOLUTION LINES.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE SHOWIN(I1)
      COMMON /AAH/ XS(120),YS(120),ZS(120),POINTM
      INTEGER POINTM
      IT=I1/2
      DO 1600 LI=1,POINTM
        YS(LI)=YS(LI)/2.0
1600   CONTINUE
      RETURN
      END

```



```

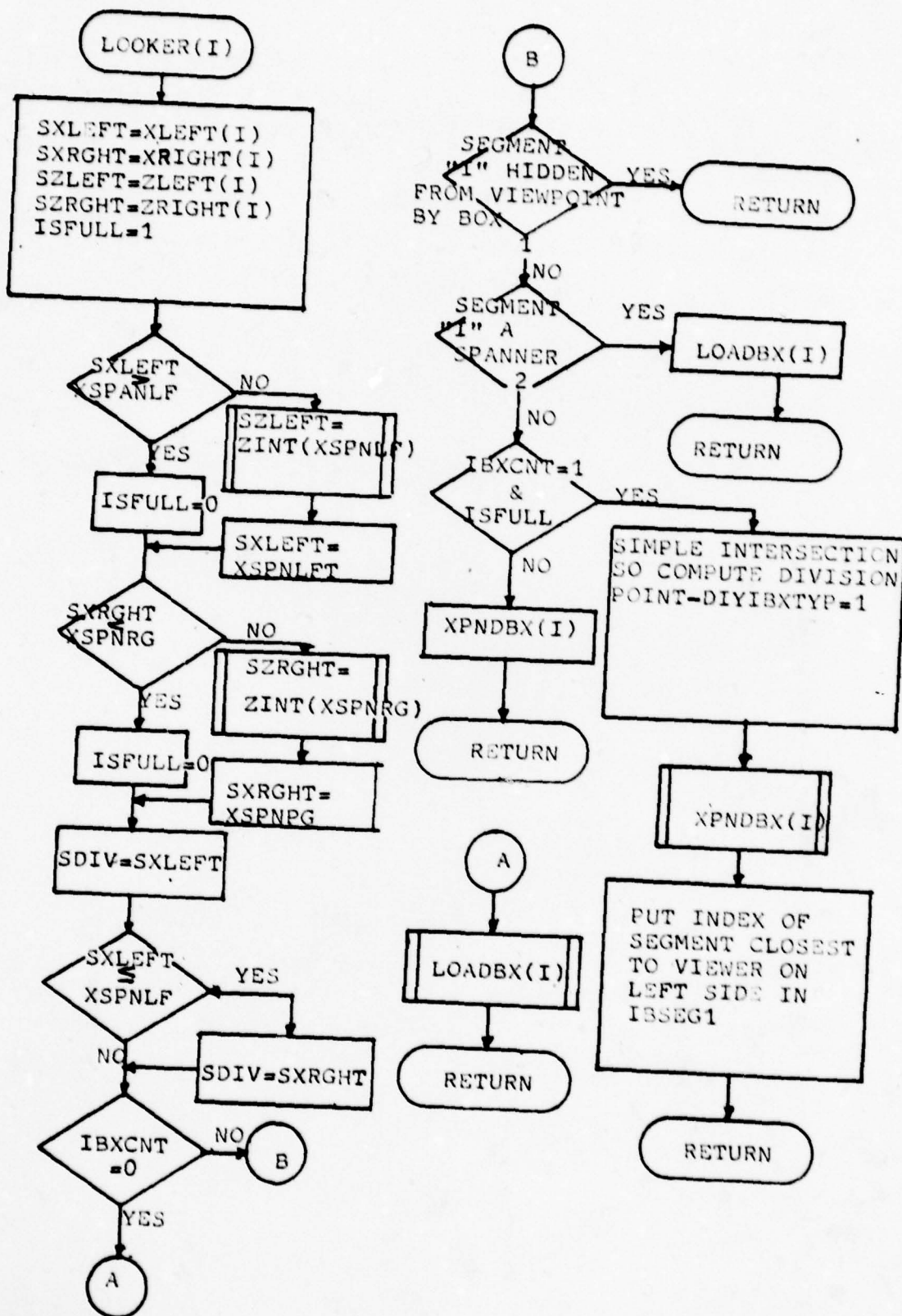
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      LOOKER: EXAMINES THE ACTIVE LIST OF SEGMENTS AND DETERMINES
C      HOW MANY ARE VIEWABLE IN THIS SPAN.  IF ALSO GENERATES
C      A BOX AROUND ALL VIEWABLE SEGMENTS AND PASSES THE
C      NUMBER OF SEGMENTS ARE IN THE BOX AND BOX TYPE TO THE
C      THINKR.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE LOOKER(I)
COMMON /TD/XLEFT(60),XRIGHT(60),ZLEFT(60),ZRIGHT(60)
COMMON /TE/XSPNLF,XSPNRG,XRESL
COMMON /IF/IBACNT,IBXTYP
COMMON /TG/BXLEFT,BXRIGHT,BZLEFT,BZRIGHT,BZMIN,BZMAX
COMMON /TH/SXLEFT,SXRIGHT,SZLEFT,SZRIGHT
COMMON /TI/ISEG1,ISEG2,DIV,SDIV,ISFULL,ISFULL
SXLEFT=XLEFT(I)
SXRIGHT=XRIGHT(I)
SZLEFT=ZLEFT(I)
SZRIGHT=ZRIGHT(I)
ISFULL=1
IF(SXLEFT.GT.XSPNLF) GO TO 611
SZLEFT=ZINT(XSPNLF)
SXLEFT=XSPNLF
GO TO 612
611 ISFULL=0
612 IF(SXRIGHT.LT.XSPNRG) GO TO 613
SZRIGHT=ZINT(XSPNRG)
SXRIGHT=XSPNRG
GO TO 614
613 ISFULL=0
614 SDIV=SXLEFT
IF(SXLEFT.LE.XSPNLF) SDIV=SXRIGHT
IF(IBACNT.NE.0) GO TO 615
CALL LOADBX(I)
RETURN
615 IF(IBACNT.NE.1) GO TO 616
Z1=BZINT(SXLEFT)
Z2=BZINT(SXRIGHT)
IF((BXLEFT.LE.SXLEFT).AND.(BXRIGHT.GE.SXRIGHT).AND.
& (Z1.LE.SZLEFT).AND.(Z2.LE.SZRIGHT)) RETURN
Z1=ZINT(BXLEFT)
Z2=ZINT(BXRIGHT)
IF(SXLEFT.GT.BXLEFT) GO TO 618
IF(SXRIGHT.LT.BXRIGHT) GO TO 618
IF(Z1.GT.BZLEFT) GO TO 618
IF(Z2.GT.BZRIGHT) GO TO 618
CALL LOADBX(I)
RETURN
618 IF(ISFULL.EQ.0) GO TO 619
IF(ISFULL.EQ.0) GO TO 619
TEMP=BXLEFT+(BXRIGHT-BXLEFT)*(SZLEFT-BZLEFT)/(BZRIGHT-
& BZLEFT-SZRIGHT+SZLEFT)
CALL APNDHX(I)
IBXTYP=1
DIV=TEMP
IF(BZLEFT.LT.SZLEFT) CALL ISWAP(ISEG1,ISEG2)
RETURN
619 CALL APNDHX(I)
RETURN
616 IF(IBACNT.LE.1) RETURN
IF(SXLEFT.GT.BXLEFT) GO TO 620
IF(SXRIGHT.LT.BXRIGHT) GO TO 620

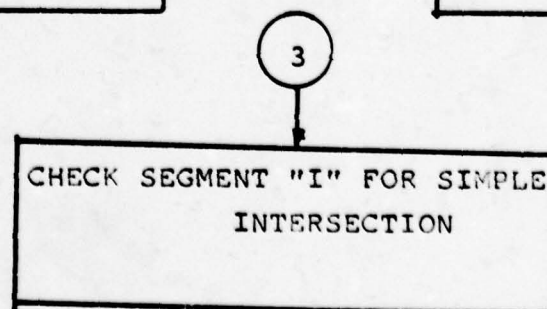
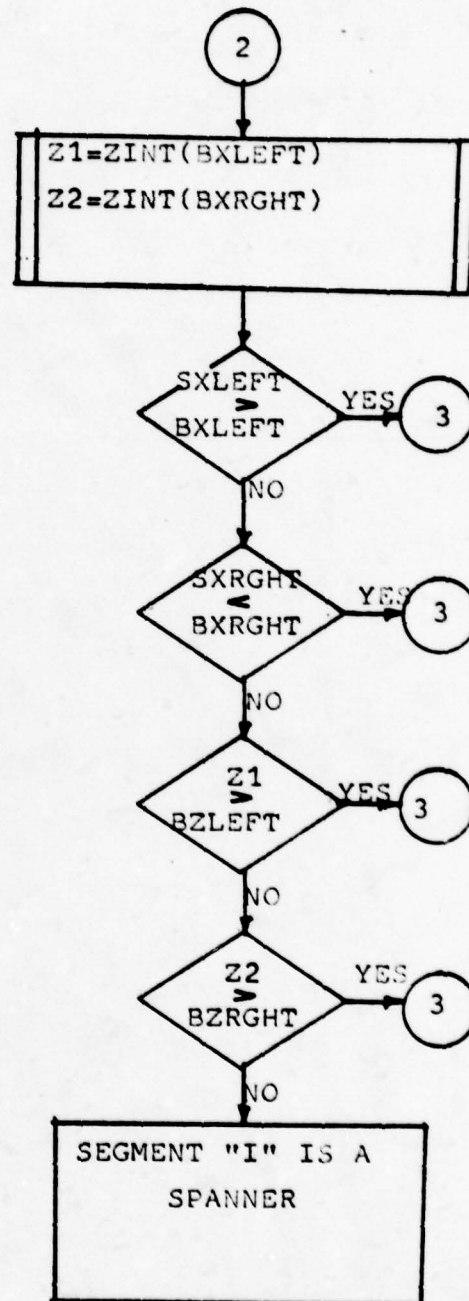
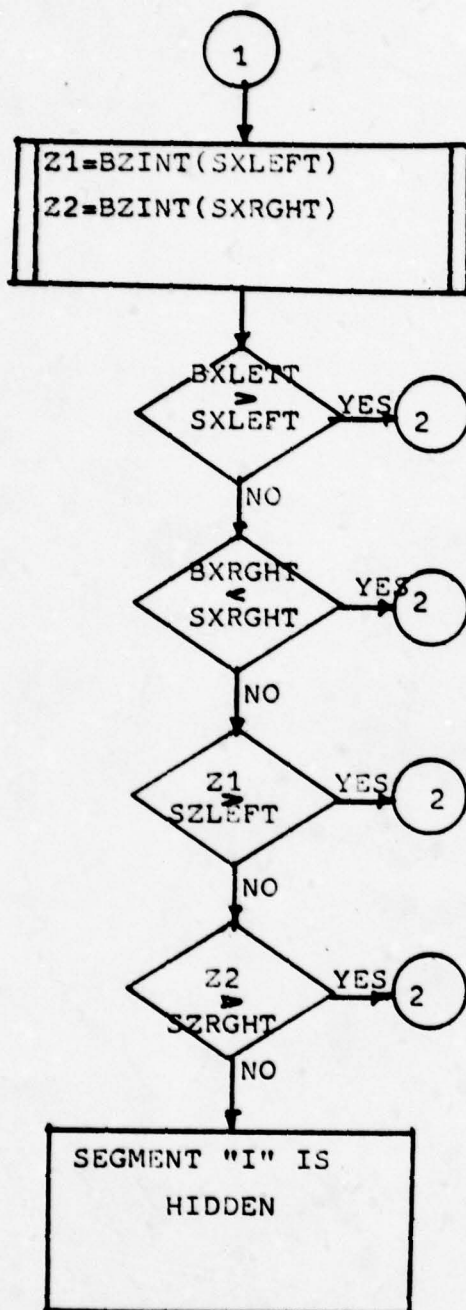
```

```

        IF(SZLEFT,G1,BZMIN) GO TO 620
        IF(SZRGHT,G1,BZMIN) GO TO 620
        CALL LOADBX(I)
        RETURN
620     IF(BXLEFT,G1,SXLEFT) GO TO 621
        IF(BXRGHT,G1,SXRGHT) GO TO 621
        IF(BZMAX,G1,SZLEFT) GO TO 621
        IF(BZMAX,G1,SZRGHT) GO TO 621
        RETURN
621     CALL APNDHX(I)
        RETURN
END

```



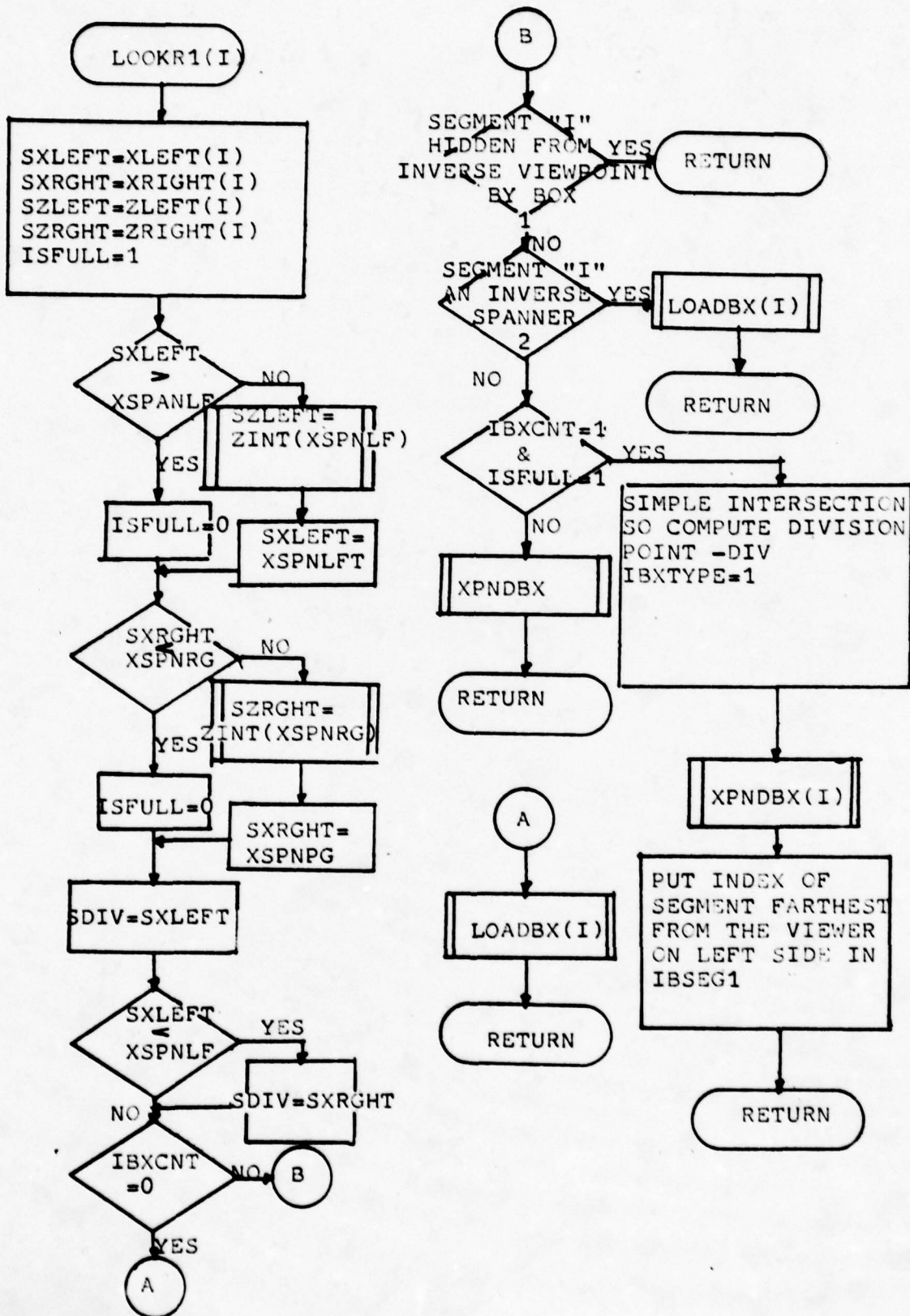


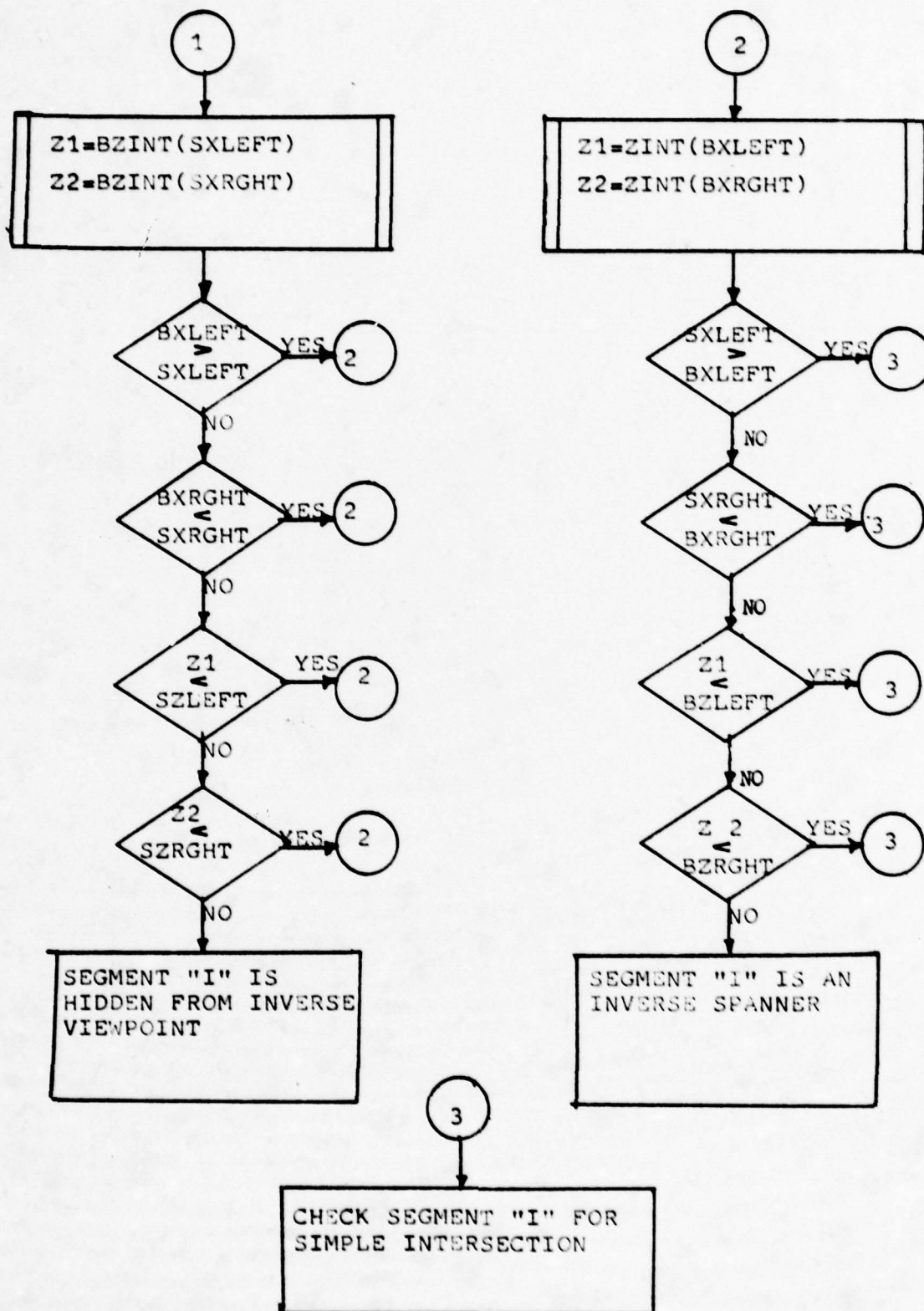


```

        IF(SZLEFT.LT.BZMAX) GO TO 620
        IF(SZRGHT.LT.BZMAX) GO TO 620
        CALL LOADBX(I)
        RETURN
620     IF(BXLEFT.GT.SXLEFT) GO TO 621
        IF(BXRGHT.LT.SXRGHT) GO TO 621
        IF(BZMIN.LT.SZLEFT) GO TO 621
        IF(BZMIN.LT.SZRGHT) GO TO 621
        RETURN
621     CALL XPNDBX(I)
        RETURN
        END

```

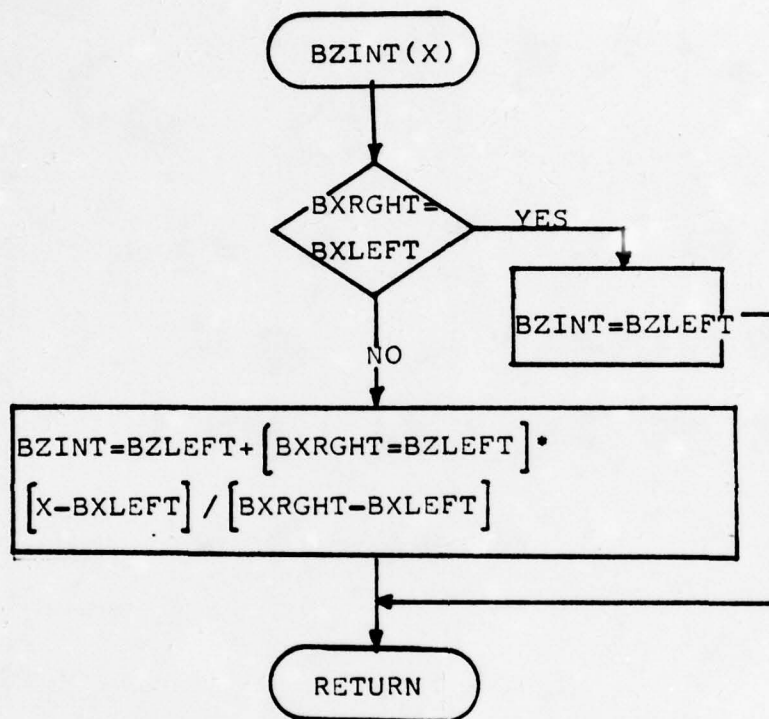




```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      BZINT: IS USED BY THE LOOKER TO DETERMINE THE DEPTH (THE ZS
C      VALUE) OF THE SEGMENT BOX AT ANY XS VALUE WITHIN THIS
C      SPAN.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION BZINT(X)
      COMMON /IG/BXLEFT,BXRIGHT,BZLEFT,BZRIGHT,BZMIN,BZMAX
      IF(BXRIGHT.EQ.BXLEFT) GO TO 621
      BZINT=BZLEFT+(BZRIGHT-BZLEFT)*(X-BXLEFT)/(BXRIGHT-BXLEFT)
      RETURN
621  BZINT=BZLEFT
      RETURN
      END

```



CC

C

C

C

C

LOADBX: IS USED BY THE LOOKER TO CONSTRUCT A BOX AROUND THE  
SEGMENTS WHICH ARE VIEWABLE IN THIS SPAN.

CC

SUBROUTINE LOADBX(I)

COMMON /TF/IBXCNT,IBXTYP

COMMON /IG/BXLEFT,BXRIGHT,BZLEFT,BZRIGHT,BZMIN,BZMAX

COMMON /TH/SXLEFT, SXRIGHT, SZLEFT, SZRIGHT

COMMON /TI/ISEG1, ISEG2, DIV, SDIV, IBFULL, ISFULL

IBXCNT=1

IBXTYP=0

BXLEFT= SXLEFT

BXRIGHT= SXRIGHT

BZLEFT= SZLEFT

BZRIGHT= SZRIGHT

ISEG1=1

BZMIN= BZLEFT

BZMAX= BZRIGHT

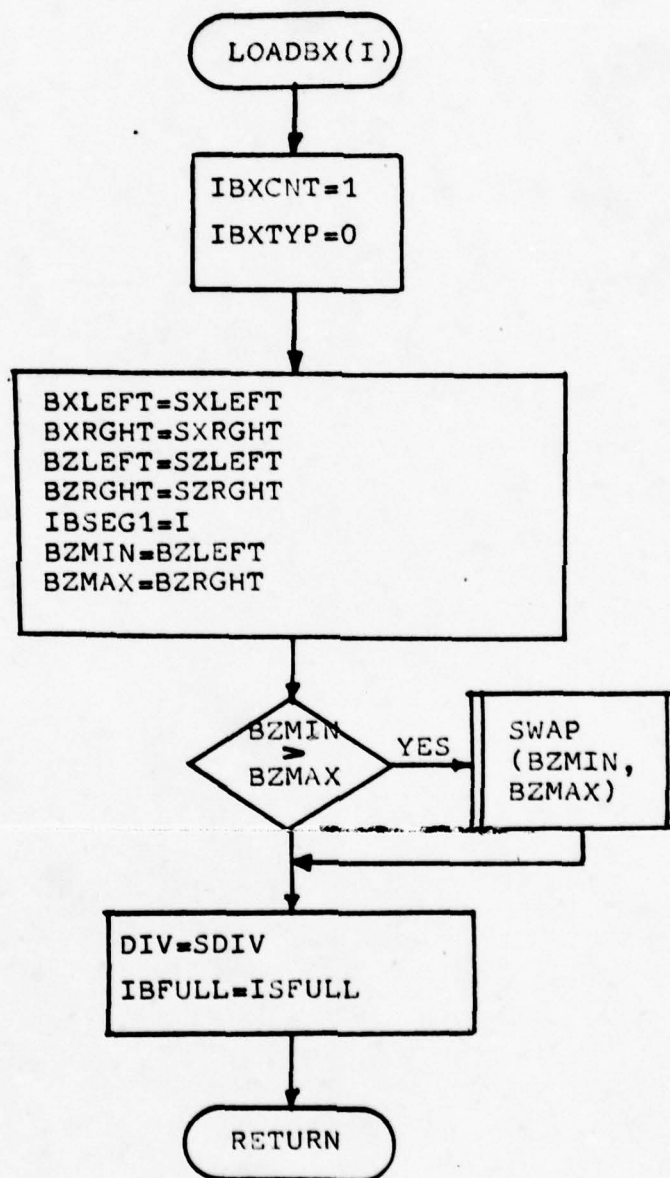
IF (BZMIN.GT.BZMAX) CALL SWAP(BZMIN,BZMAX)

DIV=SDIV

IBFULL=ISFULL

RETURN

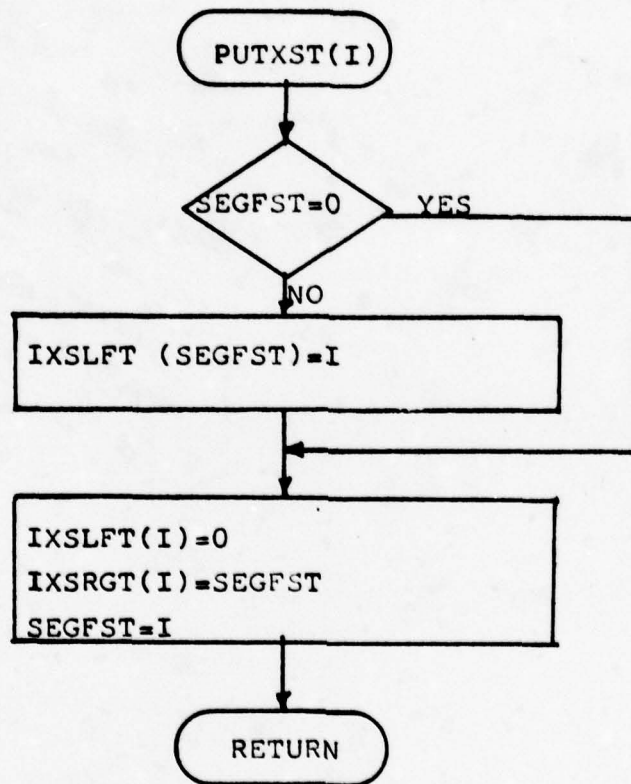
END



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      PUTXST: PLACES THE SEGMENT PASSED TO IT AS THE CALLING
C      PARAMETER I INTO OF THE XSORT LISTS, IXLFT(I)
C      AND IXRGT(I).
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE PUTXST(I)
      COMMON /IC/IXLFT(60),IXRGT(60),SEGFST
      INTEGER SEGFST
      IF(SEGFST.NE.0) IXLFT(SEGFST)=I
      IXLFT(I)=0
      IXRGT(I)=SEGFST
      SEGFST=I
      RETURN
      END

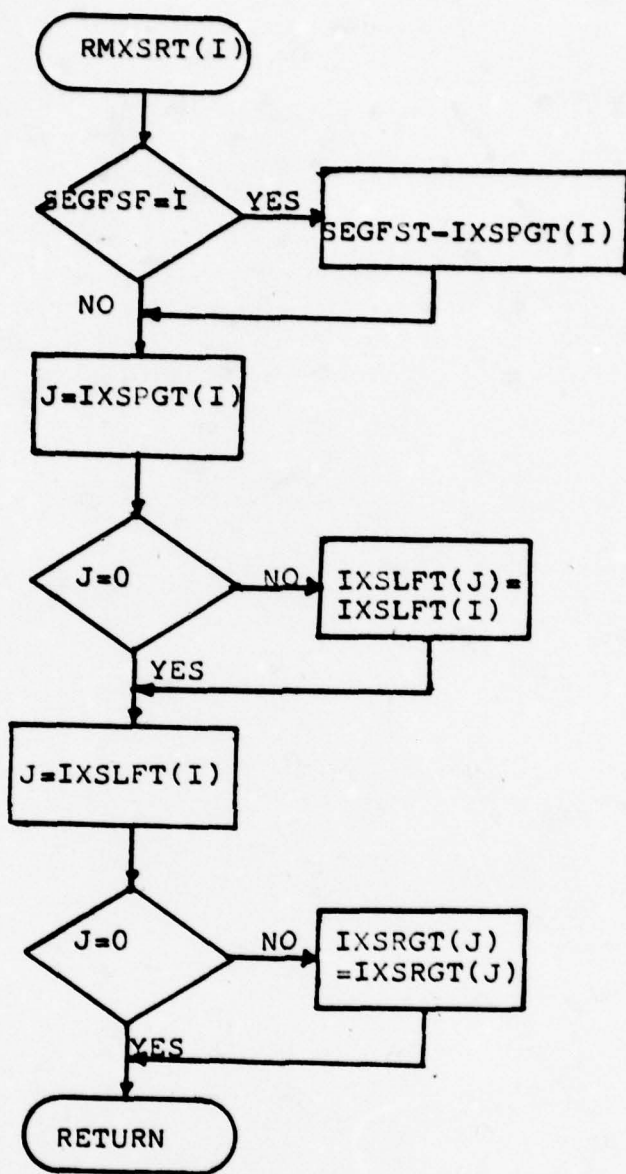
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      RMXSRT: REMOVES A SEGMENT FROM THE XSORT LISTS, IXSRGT(1)
C              AND IXSLFT(1).
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      SUBROUTINE RMXSRT(I)
C      COMMON /TC/IXSLFT(60),IXSRGT(60),SEGFST
C      INTEGER SEGFST
C      IF(SEGFST.EQ.1)SEGFST=IXSRGT(1)
C      J=IXSRGT(1)
C      IF(J.NE.0) IXSLFT(J)=IXSLFT(1)
C      J=IXSLFT(1)
C      IF(J.NE.0) IXSRGT(J)=IXSRGT(1)
C      RETURN
C      END

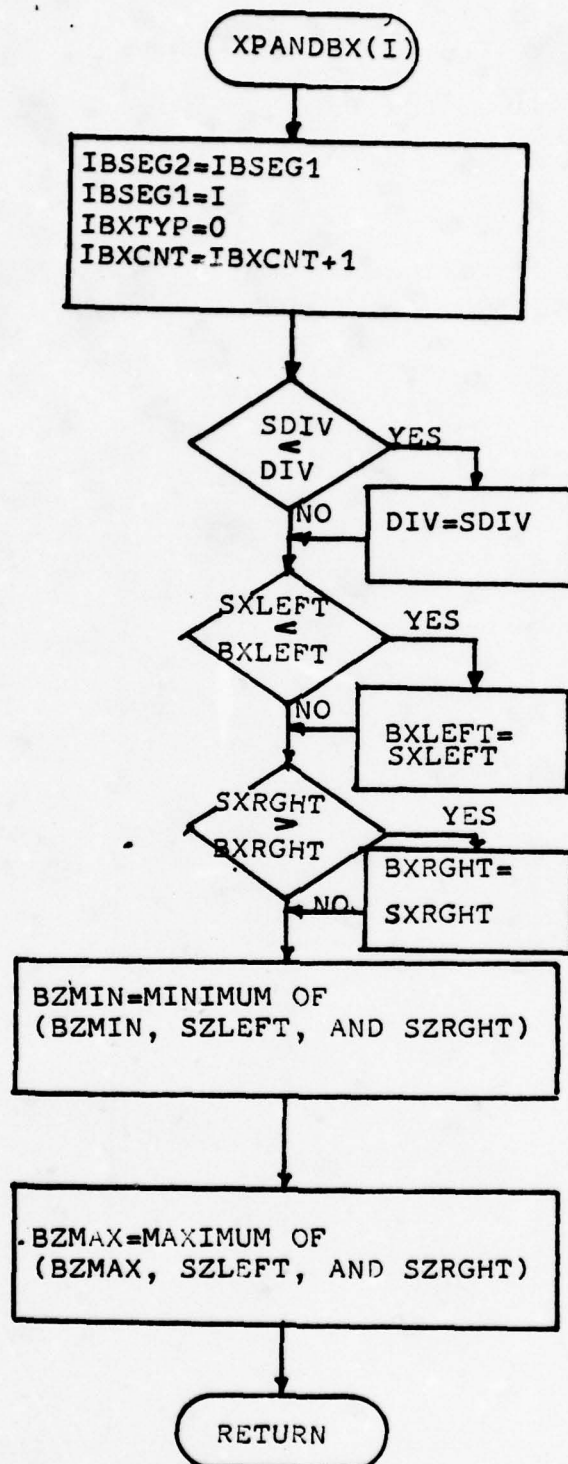
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      XPNDRX: IS CALLED BY THE LOOKER TO XPAND THE BOX SURROUNDING
C      THE VIEWABLE SEGMENTS IN THE CURRENT SPAN.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE XPNDRX(I)
COMMON /TF/IBXCNT,IBXTYP
COMMON /TG/BXLEFT,BXRGT,BZLEFT,BZRGT,BZMIN,BZMAX
COMMON /TH/SXLEFT,SXRGT,SZLEFT,SZRGT
COMMON /TI/IBSEG1,IBSEG2,DIV,SDIV,ISFULL,ISFULL
IBSEG2=IBSEG1
IBSEG1=I
IBXTYP=0
IBXCNT=IBXCNT+1
IF(SDIV.LT.DIV)DIV=SDIV
IF(SXLEFT.LT.BXLEFT) BXLEFT=SXLEFT
IF(SXRGT.GT.BXRGT) BXRGT=SXRGT
IF(SZLEFT.LT.BZMIN) BZMIN=SZLEFT
IF(SZRGT.LT.BZMIN) BZMIN=SZRGT
IF(SZLEFT.GT.BZMAX) BZMAX=SZLEFT
IF(SZRGT.GT.BZMAX) BZMAX=SZRGT
RETURN
END

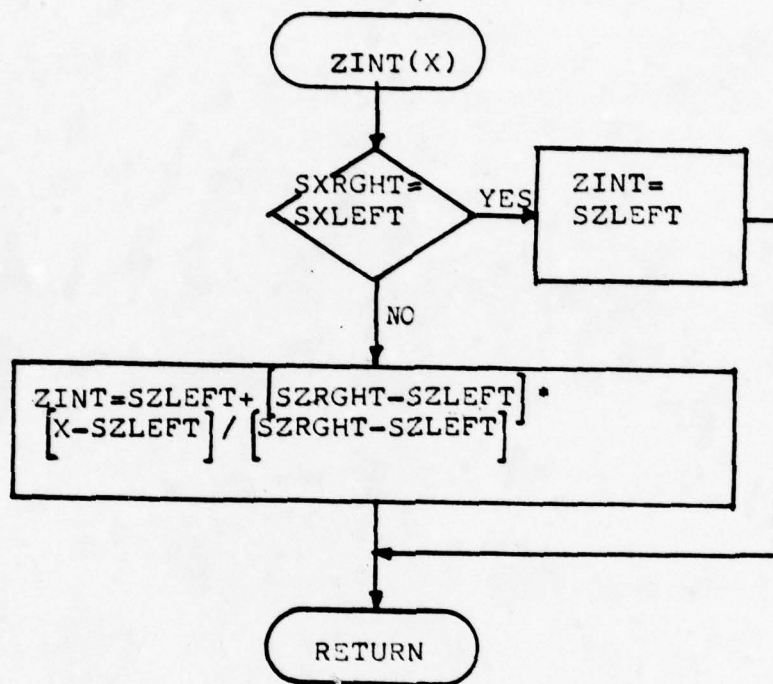
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      ZINT: USED BY THE LOOKER TO COMPUTE THE DEPTH (ZS VALUE)
C      OF THE CURRENT SEGMENT AT ANY XS VALUE, WITHIN THIS
C      SPAN.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      FUNCTION ZINT(X)
      COMMON /TH/SXLEFT, SXRGT, SZLEFT, SZRGT
      IF (SXRGT.EQ.SXLEFT) GO TO 622
      ZINT=SZLEFT+(SZRGT-SZLEFT)*(X-SXLEFT)/(SXRGT-SXLEFT)
      RETURN
622  ZINT=SZLEFT
      RETURN
      END

```



CCCCC

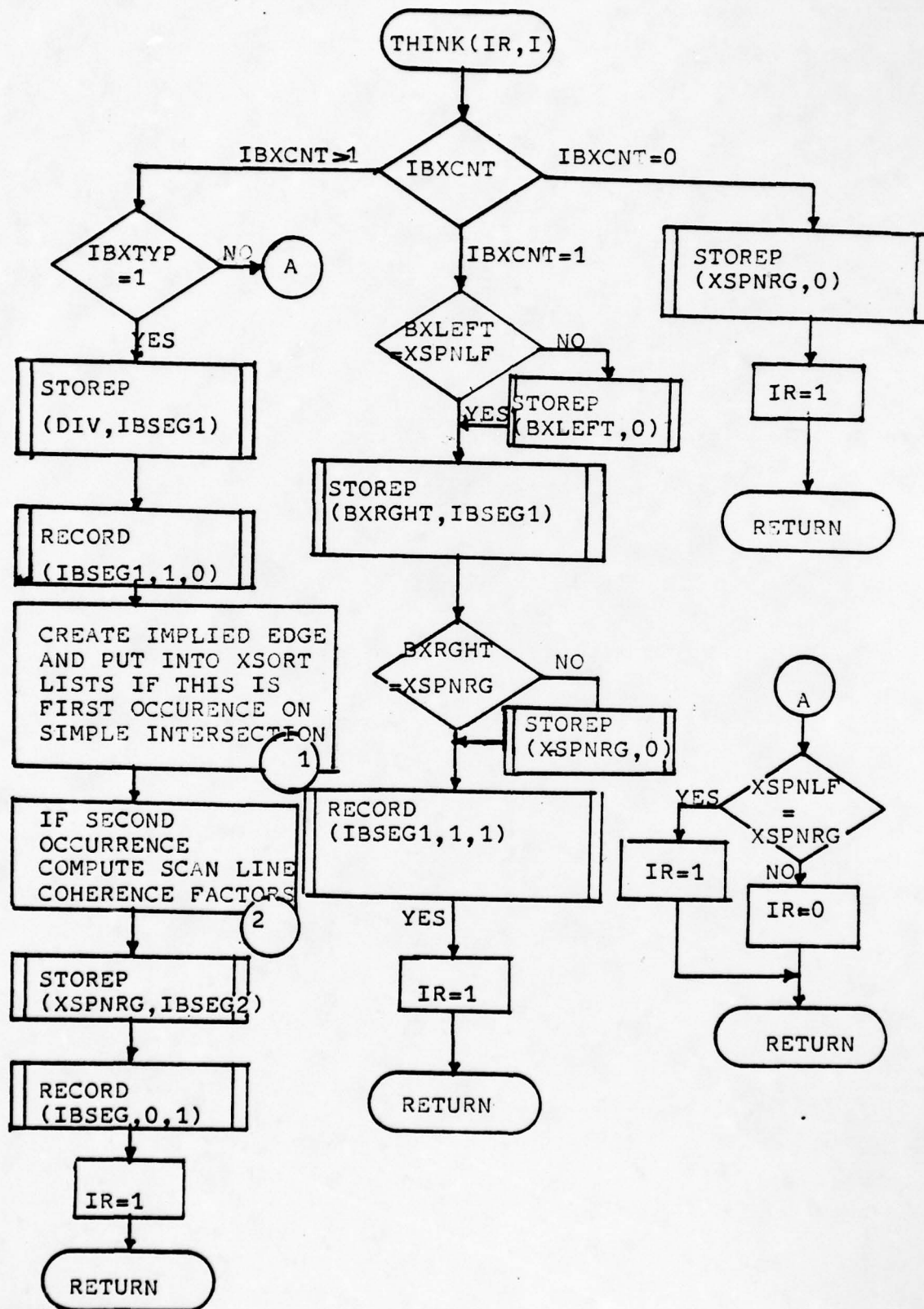
CCCCCCC

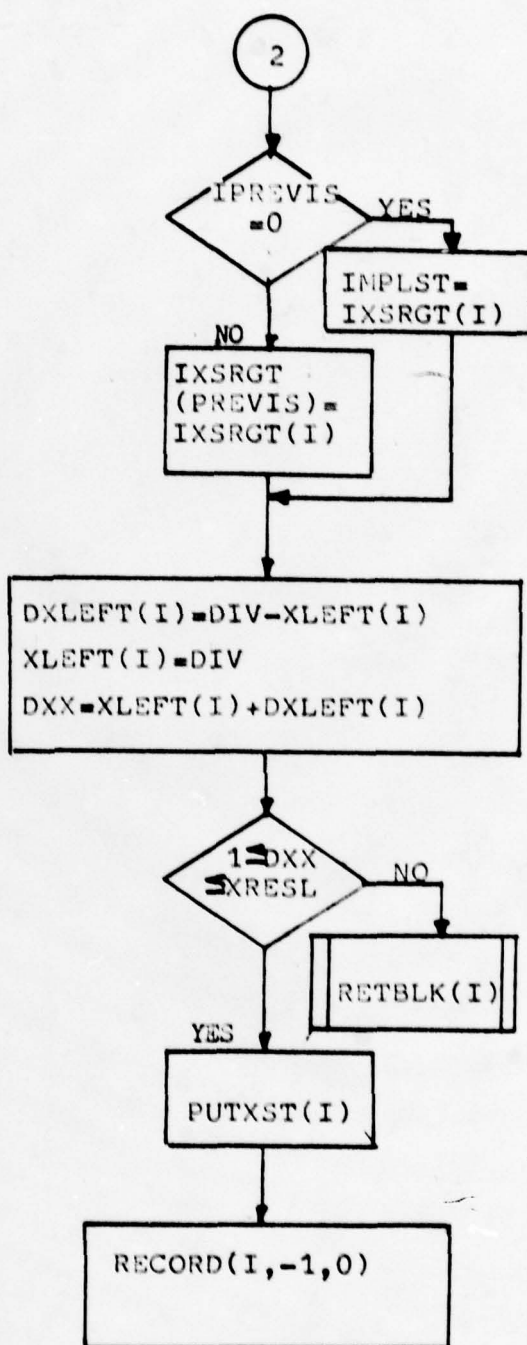
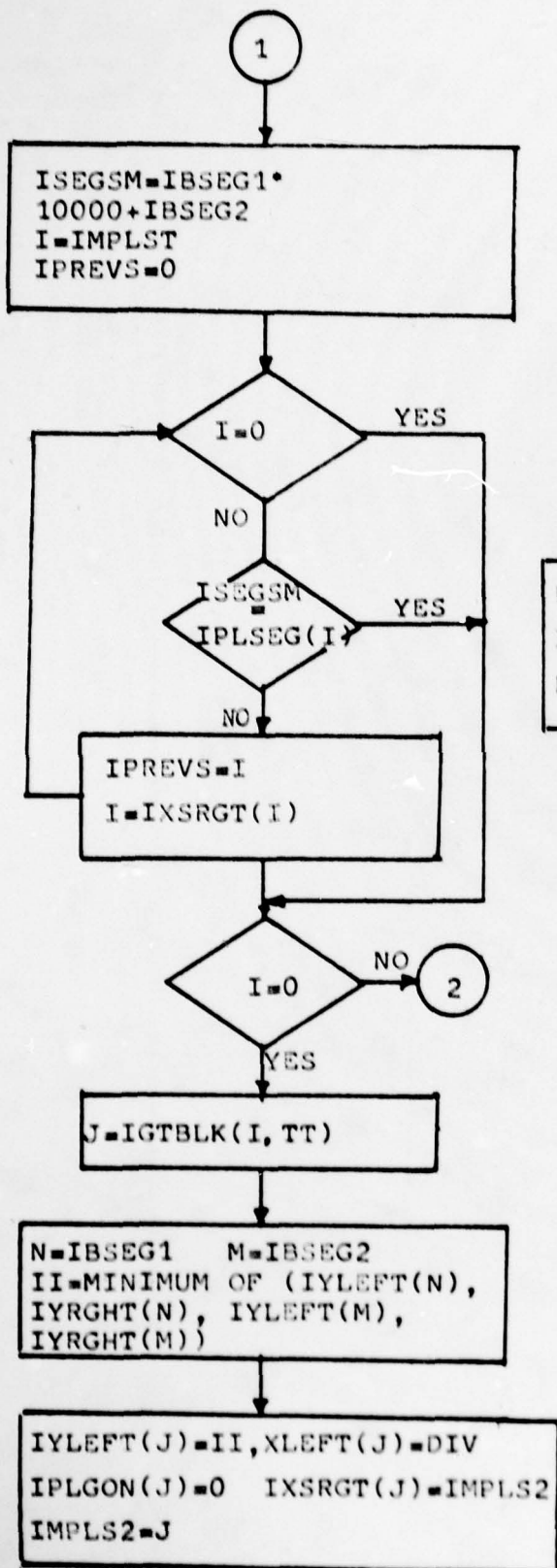
631  
628

```

        IPLSEG(J)=ISEGSM
        II=IYLEFT(1HSEG1)
        IF(II.LT.IYRGHT(1HSEG1)) II=IYRGHT(1HSEG1)
        IF(II.LT.IYLEFT(1HSEG2)) II=IYLEFT(1HSEG2)
        IF(II.LT.IYRGHT(1HSEG2)) II=IYRGHT(1HSEG2)
        IYLEFT(J)=II
        IPLGON(J)=0
        XLEFT(J)=DIV
        IXSGT(J)=IMPLS2
        IMPLS2=J
632     CALL STOREP(XSPNRG,1HSEG2)
        CALL RECORD(1HSEG2,1ZERO,1ONE)
        IR=1
        RETURN
625     IF(XSPNLF.EQ.XSPNRG) GO TO 633
        IR=0
        RETURN
633     IR=1
        RETURN
160    WRITE(6,999)
999    FORMAT(2X,'THE NUMBER OF SEGMENTS HAS EXCEEDED THE STORAGE
&PROVIDED')
        IR=2
        END

```





```

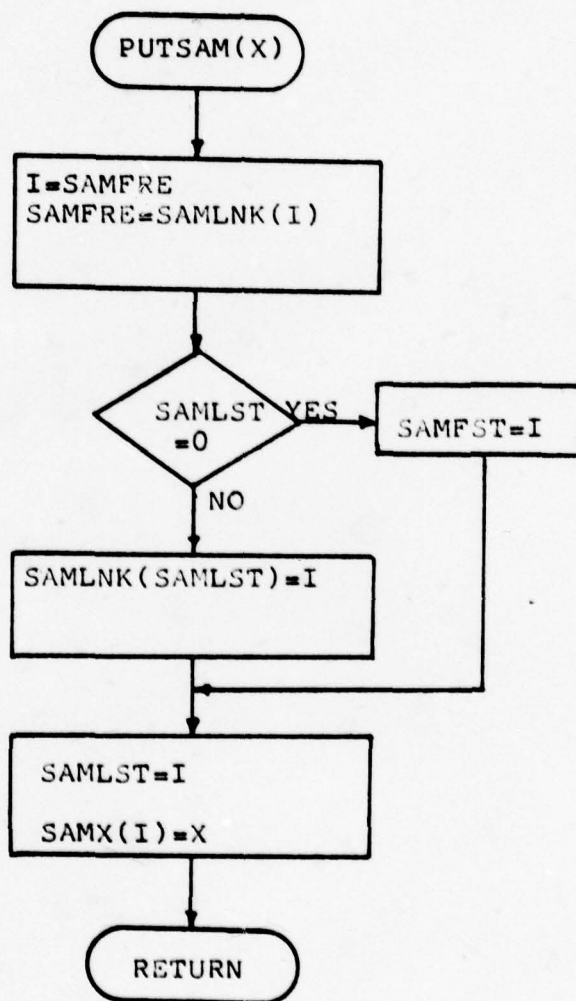
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      PUTSAM: STORES THE SUCCESSFUL SAMPLE POINTS (THE SCAN LINE
C      DIVISION POINTS) OF THE PRESENT SCAN LINE WHICH WILL
C      BE USED TO DIVIDE THE NEXT SCAN LINE. THIS REDUCES THE
C      TIME REQUIRED TO PROCESS AND DISPLAY AN IMAGE. THIS
C      PROCEDURE IS CALLED BY SUBROUTINE RECORD.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

      SUBROUTINE PUTSAM(X)
      COMMON /TM/SAMFRE,SAMLST,SAMLNK(120),SAMX(120)
      INTEGER SAMFRE,SAMLST,SAMLNK,SAMX
      I=SAMFRE
      SAMFRE=SAMLNK(I)
      IF(SAMLST.EQ.0) GO TO 634
      SAMLNK(SAMLST)=I
      GO TO 635
634  SAMFST=I
635  SAMLST=I
      SAMX(I)=X
      RETURN
      END

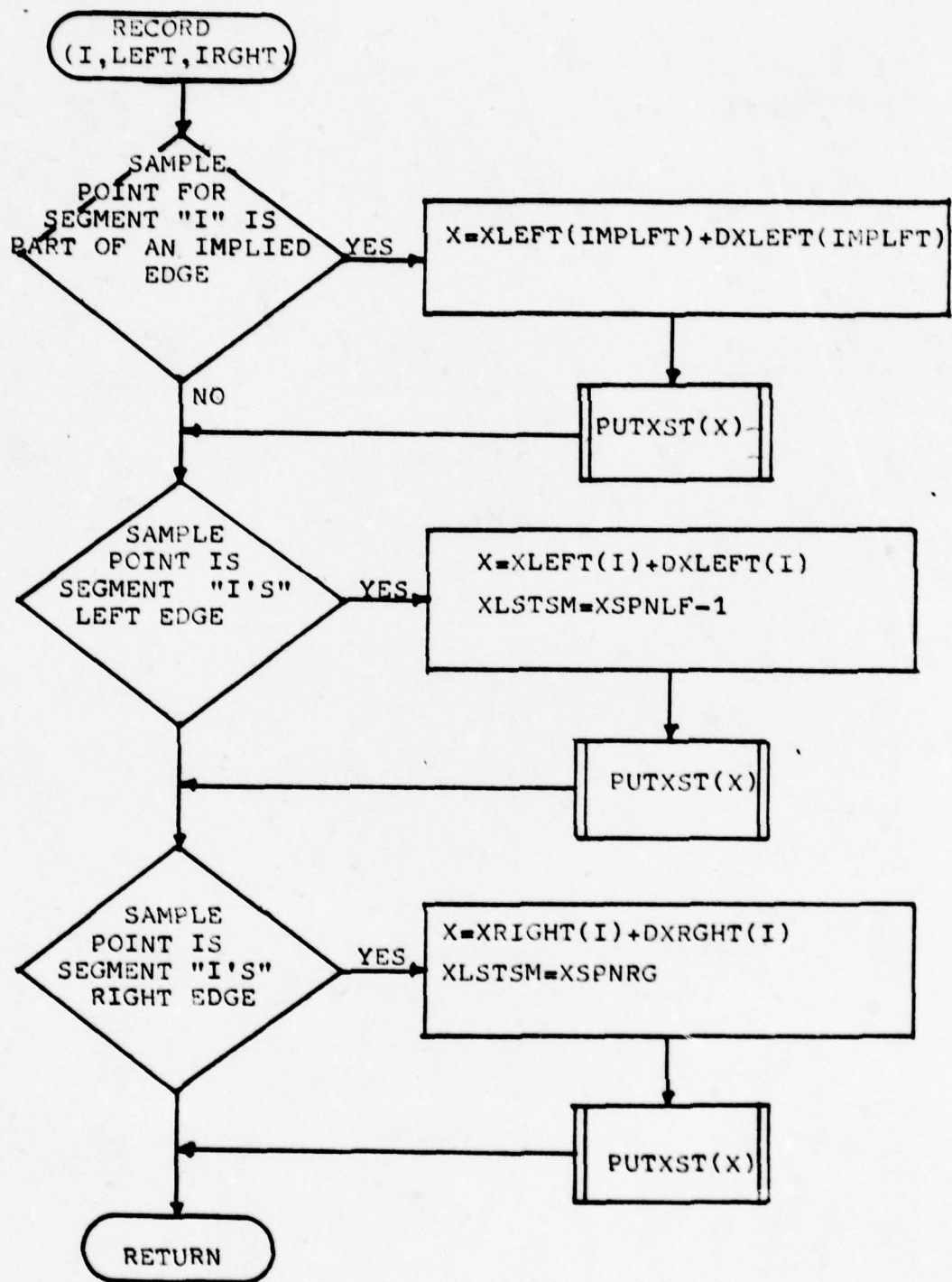
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      RECORD: DECIDES WHICH SCAN LINE SAMPLE POINTS SHOULD BE
C      SAVED TO DIVIDE THE NEXT SCAN LINE INTO SPANS.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE RECORD(I,LEFT,IRGHT)
      COMMON /TN/IMPLFT,XLSISM
      COMMON /TU/XLEFT(60),XRIGHT(60),ZLEFT(60),ZRIGHT(60)
      COMMON /TK/POLSEG(60),POLGON(60)
      COMMON /TL/DXLEFT(60),DXRGHT(60)
      COMMON /TB/IYLEFT(60),IYRGHT(60)
      COMMON /TM/ SAMFRE,SAMLST,SAMLNK(120),SAMX(120)
      COMMON /TE/ XSPNLF,XSPNRG,XRESL
      INTEGER POLSEG,SAMFRE,SAMLST,SAMLNK,SAMX,POLGON
      IF(LEFT.EQ.0) GO TO 637
      IF(IMPLFT.EQ.0) GO TO 636
      IF(XLEFT(1).GT.XSPNLF) GO TO 636
      J=POLSEG(IMPLFT)/10000
      II=POLSEG(IMPLFT)-J*10000
      IF(I.NE.II) GO TO 636
      X=XLEFT(IMPLFT)+DXLEFT(IMPLFT)
      CALL PUTSAM(X)
      IMPLFT=0
636  IF(IYLEFT(1).GE.-1)GO TO 637
      DEL=XSPNLF-1.0
      XL=XLEFT(1)
      IF((LEFT.NE.-1).AND.((XL.LE.DEL).OR.(XL.GT.XSPNLF))) GO TO 637
      IF((SAMLST.NE.0).AND.(XLSISM.EQ.DEL).AND.(LEFT.NE.-1))
      & GO TO 637
      X=XLEFT(1)+DXLEFT(1)
      CALL PUTSAM(X)
      XLSISM=XSPNLF-1.0
637  IF(IRGHT.EQ.0) RETURN
      IF(IYRGHT(1).GE.-1) RETURN
      IF(XSPNRG.GT.XRIGHT(1)) RETURN
      UP=XSPNRG+1
      IF(XRIGHT(1).GE.UP) RETURN
      X=XRIGHT(1)+DXRGHT(1)
      CALL PUTSAM(X)
      XLSISM=XSPNRG
      RETURN
      END

```



CC

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

STOREP: USED BY THE THINKR TO RECORD THE DISPLAY DATA  
FOR THE CURRENT SCAN LINE. THE XS VALUES ARE  
STORED IN THE ARRAY ISPOS(1) AND THE SEGMENT'S  
INDEX IS STORED IN ISSEG(1).

CC

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

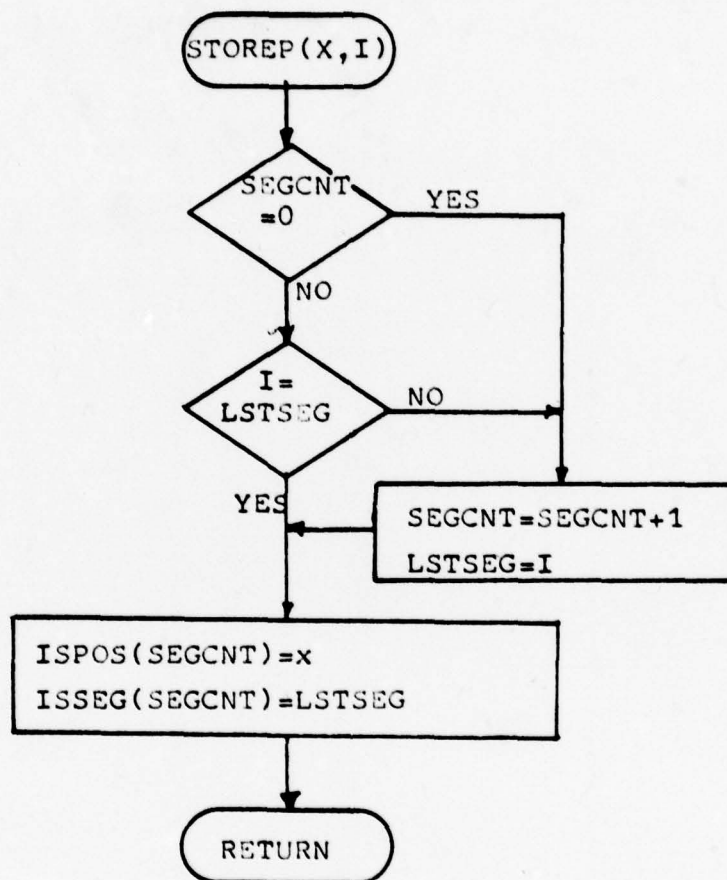
C

C

C

C

SUBROUTINE STOREP(X,I)  
COMMON /TO/SEGNT,LSTSEG  
COMMON /TP/ISPOS(60),ISSEG(60)  
INTEGER SEGNT  
IF((SEGNT.NE.0).AND.(I.EQ.LSTSEG)) GO TO 640  
SEGNT=SEGNT+1  
LSTSEG=I  
640 ISPOS(SEGNT)=X  
ISSEG(SEGNT)=LSTSEG  
RETURN  
END



## APPENDIX B

### APPLICATIONS PROGRAM AND SUBROUTINES

The applications program and its flow chart were presented first. Next, the seven subroutines which enabled the user to determine polygonal penetration, change the coordinate values of a polygon, and alter the shade or color of a polygon were listed. These programs and their flow charts were not included with the 3-D graphics software package since their applicability was strictly related to a tracking presentation.

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      THIS IS AN APPLICATION PROGRAM DESIGNED TO DEMONSTRATE THE
C      USAGE OF THE THREE-DIMENSIONAL COMPUTER GRAPHICS SOFTWARE
C      PACKAGE. THE PROGRAM SIMULATES THE TRACKING OF A TORPEDO IN AN
C      IRREGULAR TORPEDO TEST AREA.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      DIMENSION IR(2),S(3),IO(3),I(3),TRACKX(5),TRACKY(5),TRACKZ(5),
C      &P2(3),DVECTO(3),N(3),P1(3)
C      DATA DVECTO/0.0,-1.0,0.0/
C      CALL INITAL
C      CONV=180.0/3.14159
C      IRED=N
C      IPH=5
C      ISHAD=16
C      NM=0
C      I=1
C      IR(2)=5
C      RISUM=0.0
C
C      INITIALIZING THE OLD DIRECTION VECTOR
C
C      DO 80 J=1,3
C      80 RISUM=RISUM+DVECTO(J)**2
C
C      MUST SCALE THE IMAGE OF THE TORPEDO SO THAT IT IS VISIBLE
C
C      IR(1)=4
C      IR(2)=4
C      P2(1)=10.0
C      P2(2)=10.0
C      P2(3)=10.0
C      CALL SCALE(IR,P2)
C
C      ROTATE THE VEHICLE SO THAT IT IS INITIALLY BROADSIDE TO THE
C      VIEWER.
C
C      THETA=90.0
C      IAXIS=2
C      CALL ROTATE(IR,IAXIS,P1,P2,THETA)
C      READ(5,1) 1
C      DO 10 J=1,3
C      10 S(J)=-I(J)
C      IAXIS=3
C
C      BEGIN THE RECURSIVE PORTION OF THE PROGRAM
C
C      200 TRACKX(1)=I(1)
C      TRACKY(1)=I(2)
C      TRACKZ(1)=I(3)
C      DO 20 J=1,3
C      20 IO(J)=I(J)
C
C      DISPLAY THE TORPEDO TEST AREA WITH HIDDEN SURFACES SHOWN.
C
C      ILOOK=2
C      IR(1)=1
C      IR(2)=3
C      CALL SURFAC(IR,ILOOK)
C
C      TRANSLATE THE TORPEDO TO ITS PRESENT LOCATION.
C

```

```

      ILOOK=1
      IR(1)=4
      CALL TRANSL(IR,S)
C
C      CHECK THE CURRENT LOCATION OF THE TORPEDO TO
C      DETERMINE IF IT IS WITHIN THE TEST AREA.
C
      IMP=IPENET(IR,T)
      IF(IMP.EQ.1) GO TO 220
      IF(IMP.EQ.0.AND.IFLAG.EQ.1) GO TO 240
230 IF(NM.LT.5) NM=NM+1
C
C      IF THIS AT LEAST THE SECOND LOCATION OF THE TORPEDO
C      THEN DISPLAY THE TRACK.
C
      IF(NM.GT.1) GO TO 250
C
C      DISPLAY THE TORPEDO WITH THE HIDDEN SURFACES REMOVED.
C
      IR(1)=4
      IR(2)=4
      CALL SURFAC(IR,ILOOK)
C
C      GET THE TORPEDO'S NEW LOCATION.
C
      READ(S,1) I
      I=I+1
      IF(I(1).LT.-9999.0) STOP
C
C      NOW, COMPUTE THE TARGET ASPECT OF THE TORPEDO.
C
      RISUM=0.0
      R2SUM=0.0
      DO 30 J=1,3
        S(J)=TO(J)-I(J)
        R2SUM=R2SUM+S(J)**2
30 CONTINUE
      R1=SQRT(RISUM)
      R2=SQRT(R2SUM)
      CTHETA=0.0
      DO 40 J=1,3
40 CTHETA=CTHETA+S(J)*DVECTO(J)
      THETA=ARCOS(CTHETA/(R1*R2))
      THEIA=THEIA*CONV
      N(1)=DVECTO(2)*S(3)-DVECTO(3)*S(2)
      N(2)=DVECTO(1)*S(3)-DVECTO(3)*S(1)
      N(3)=DVECTO(1)*S(2)-DVECTO(2)*S(1)
      DO 50 J=1,3
50 P2(J)=T(J)+N(J)
C
C      ROTATE THE TORPEDO ABOUT THE ARBITRARY AXIS SPECIFIED BY
C      THE TWO POINTS P2() AND S().
C
      CALL ROTATE(IR,IAXIS,1,P2,THETA)
C
C      UPDATE THE OLD DIRECTION VECTOR.
C
      RISUM=R2SUM
      DO 60 J=1,3
60 DVECTO(J)=S(J)
      GO TO 200
C
C      DISPLAY THE TRACK.

```

```

C
250 IZ=1
    IP=49
    DO 70 J=1,NM
        P1(1)=TRACKX(IZ)
        P1(2)=TRACKY(IZ)
        P1(3)=TRACKZ(IZ)
        IZ=IZ-1
        IF (IZ.LT.1) IZ=5
        CALL PUTSIN(IP,J,P1)
70 CONTINUE
    IR(1)=IPH
    IR(2)=IPH
    CALL DISPLY(IR)
    GO TO 230

```

```

C
C     THE TORPEDO HAS PENETRATED THE TEST AREA CHANGE ITS COLOR
C     AND THE TRACK'S COLOR TO RED.
C

```

```

220 IFLAG=1
    CALL CSHAD(IR,IPED)
    GO TO 230

```

```

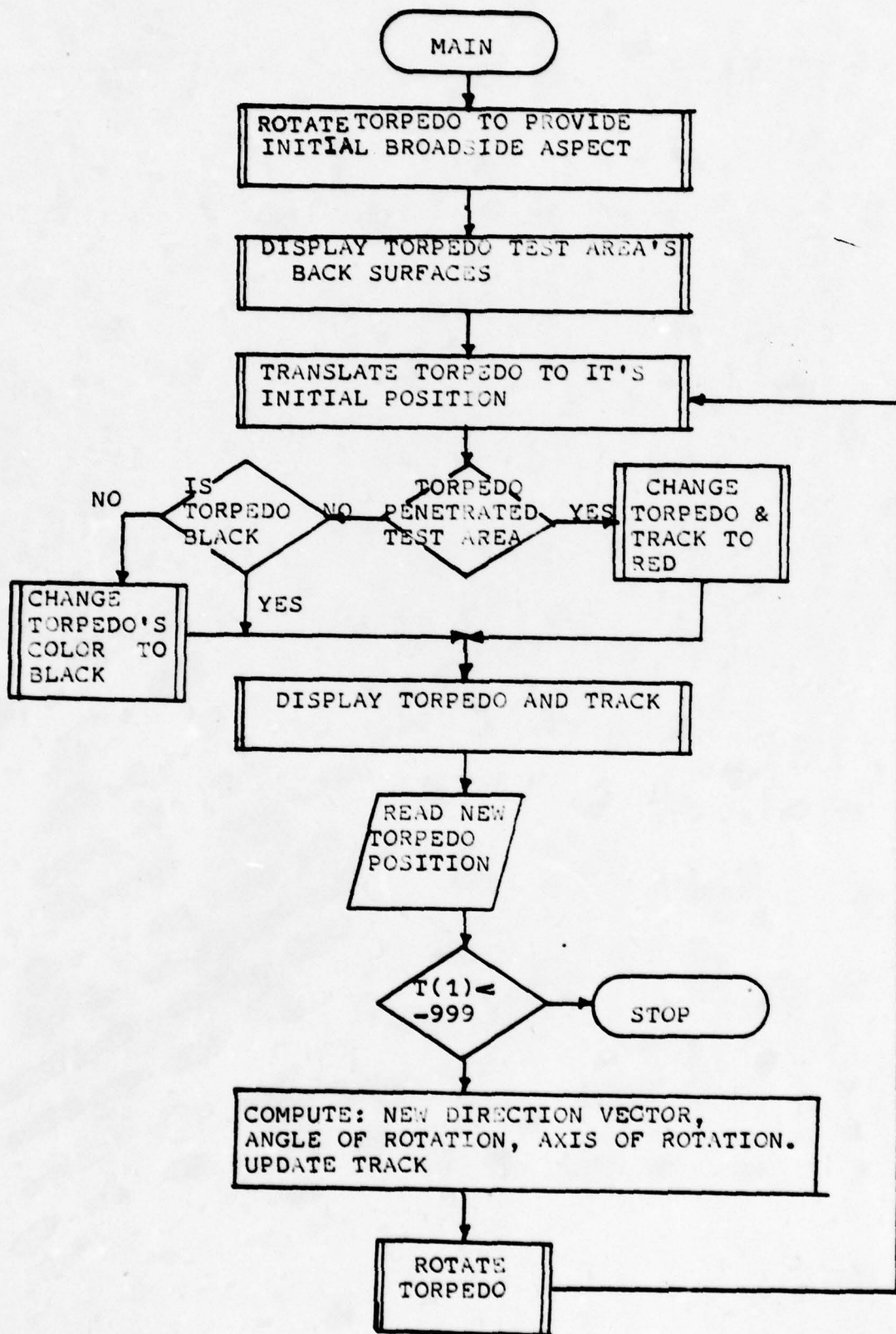
C
C     TORPEDO IS BACK IN THE TEST AREA SO CHANGE ITS COLOR TO BLACK.
C

```

```

240 IFLAG=0
    CALL CSHAD(IP,ISHAD)
    GO TO 230
1  FORMAT(3G10.3)
END

```



CC

C

C

C

C

CHGSGN: CHANGES THE SIGN OF THE COEFFICIENTS FOR THE  
POLYGON K.

CC

SUBROUTINE CHGSGN(K)

COMMON /CA/ POLYA(60),POLYB(60),POLYC(60),POLYD(60)

POLYA(K)=-POLYA(K)

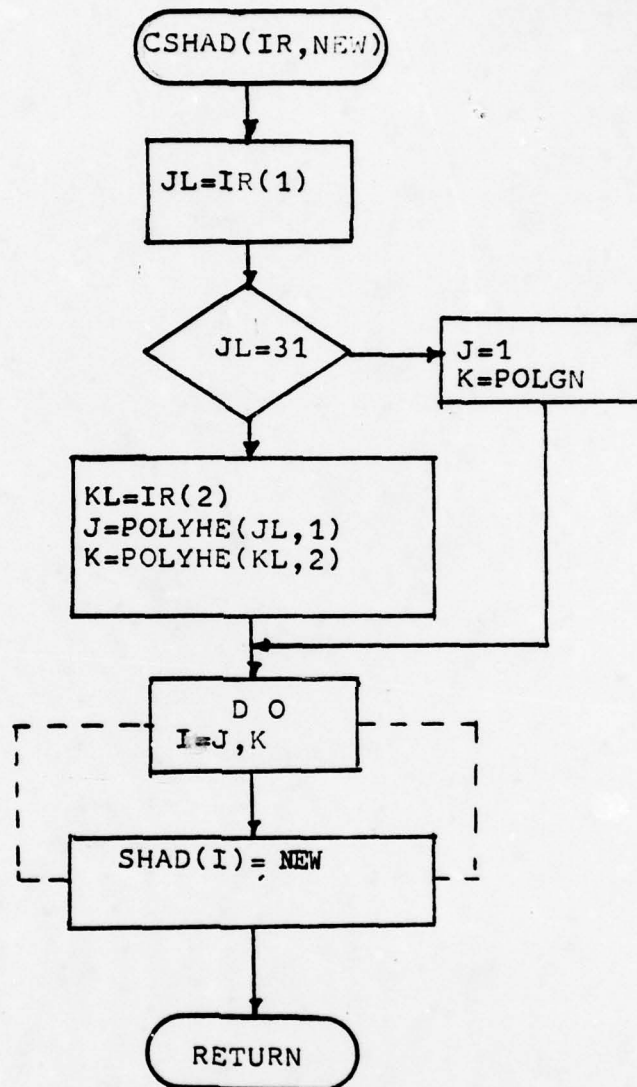
POLYB(K)=-POLYB(K)

POLYC(K)=-POLYC(K)

POLYD(K)=-POLYD(K)

RETURN

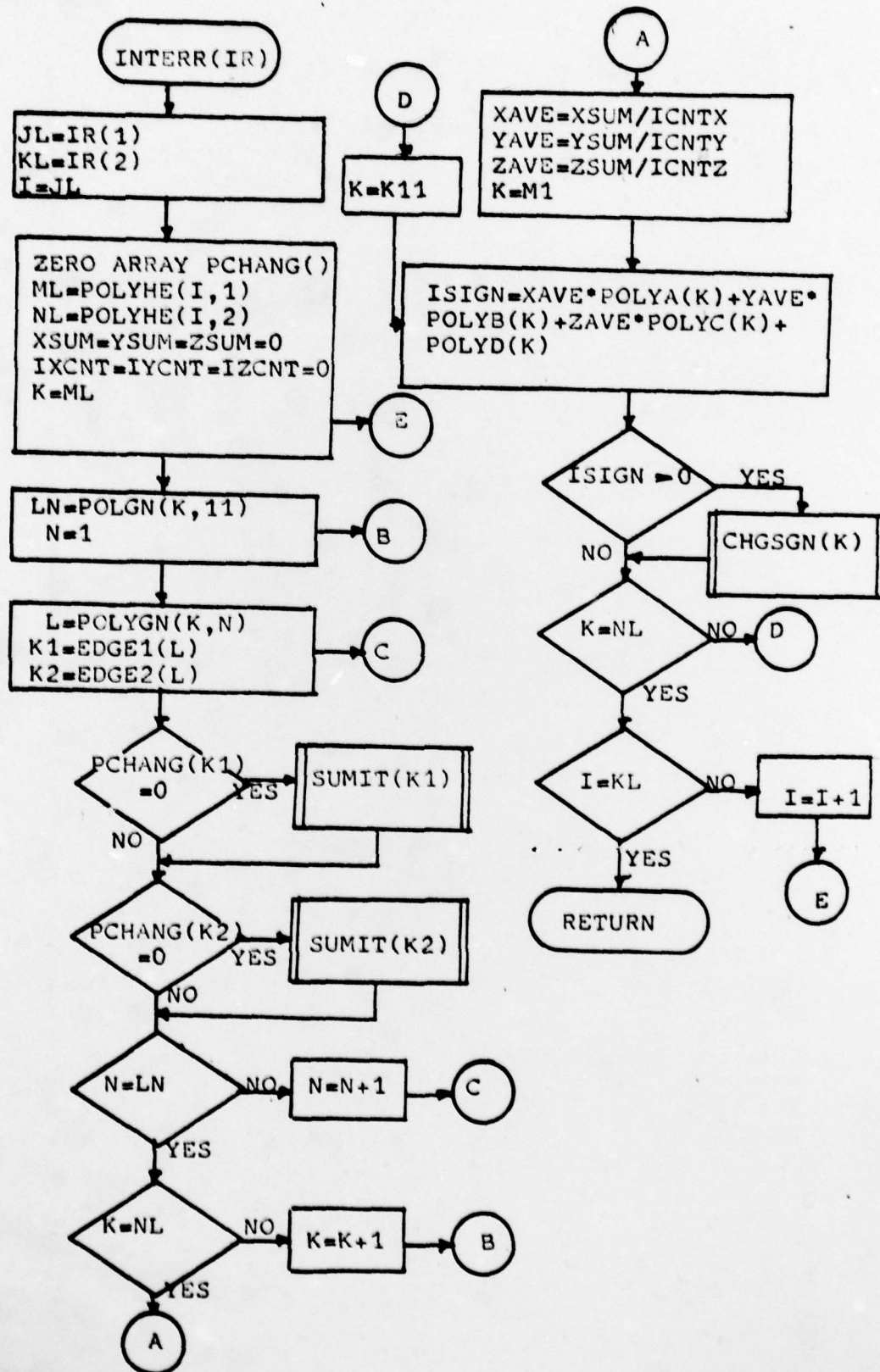
END



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      INTERR: COMPUTES THE POINT WHICH IS THE GEOMETRIC CENTER
C      OF A SET OF CONVEX POLYHEDRA. THEN, IT CHANGES THE
C      SIGN OF THE COEFFICIENTS FOR ALL POLYGONS OF EACH
C      POLYHEDRON SO THAT THE DOT PRODUCT OF AN INTERIOR
C      POINT WITH THE VECTOR NORMAL WILL BE LESS THAN ZERO.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE INTERR(IR)
      DIMENSION IR(2)
      COMMON /CA/ POLYA(60),POLYB(60),POLYC(60),POLYD(60)
      COMMON /CB/ XSUM,YSUM,ZSUM,ICNT
      COMMON /AA/ POLYHE(10,2),POLYHN
      COMMON /AB/ POLYGN(60,11),POLGN,SHAD(60)
      COMMON /AC/ EDGE1(60),EDGE2(60),EDGE3
      COMMON /FE/ PCHANG(200)
      INTEGER POLYHE,POLYHN,POLYGN,POLGN,SHAD,EDGE1,EDGE2,EDGE3,
&PCHANG
      DO 30 I=1,200
130 PCHANG(I)=0
      JL=IR(1)
      KL=IR(2)
      DO 1200 I=JL,KL
          ML=POLYHE(I,1)
          NL=POLYHE(I,2)
          XSUM=0.0
          YSUM=0.0
          ZSUM=0.0
          ICNT=0
          DO 1210 K=NL,NL
              LN=POLYGN(K,11)
              DO 1220 N=1,LN
                  L=POLYGN(K,N)
                  K1=EDGE1(L)
                  K2=EDGE2(L)
                  IF(PCHANG(K1).EQ.0) CALL SUMMIT(K1)
                  IF(PCHANG(K2).EQ.0) CALL SUMMIT(K2)
1220          CONTINUE
1210          CONTINUE
          C=ICNT
          XAVE=XSUM/C
          YAVE=YSUM/C
          ZAVE=ZSUM/C
          DO 1230 K=NL,NL
              ISIGN=XAVE*POLYA(K)+YAVE*POLYB(K)+ZAVE*POLYC(K)+POLYD(K)
              IF(ISIGN.GT.0) CALL CHGSGN(K)
1230          CONTINUE
1200          CONTINUE
      RETURN
      END

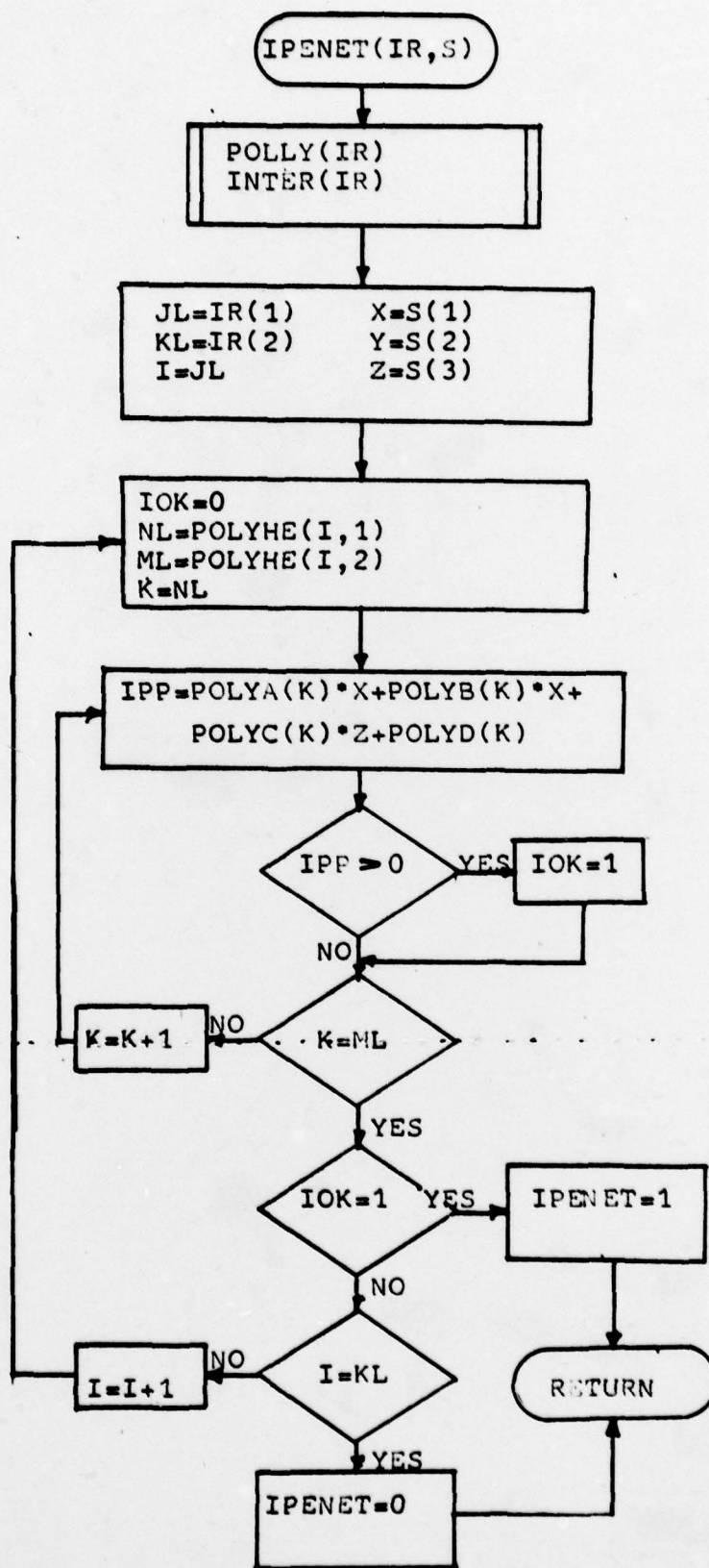
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      IPENET: DETERMINES IF A POINT IS OUTSIDE OF A GIVEN SET OF
C      POLYHEDRA. IF IT IS OUTSIDE OF THE POLYHEDRONS THIS
C      FUNCTION RETURNS A VALUE OF 1. THIS FUNCTION MUST BE
C      USED IN CONJUNCTION WITH SUBROUTINES POLLY AND INTERR.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
FUNCTION IPENET(IR,S)
  DIMENSION S(3),IP(2)
  COMMON /CA/ POLYA(60),POLYB(60),POLYC(60),POLYD(60)
  COMMON /AA/ POLYHE(10,2),POLYHN
  INTEGER POLYHE,POLYHN
  CALL POLLY(IR)
  CALL INTERR(IR)
  JL=IR(1)
  KL=IR(2)
  INSIDE=0
  X=S(1)
  Y=S(2)
  Z=S(3)
  DO 1800 I=JL,KL
    NL=POLYHE(I,1)
    ML=POLYHE(I,2)
    IOK=0
    DO 1810 K=NL,ML
      IPP=POLYA(K)*X+POLYB(K)*Y+POLYC(K)*Z+POLYD(K)
      IF(IPP.GT.0) IOK=1
1810  CONTINUE
      IF(IOK.EQ.0) INSIDE=1
1800  CONTINUE
  IPENET=0
  IF(INSIDE.EQ.0) IPENET=1
  RETURN
END

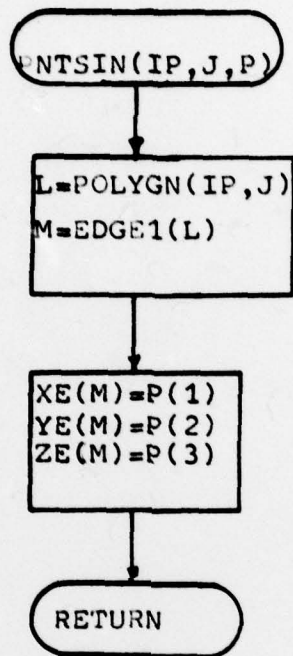
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      PNTSIN: ALLOWS THE USER TO REPLACE ANY VERTX OR POINT OF
C      A POLYGON.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE PNTSIN(IP,J,P)
  DIMENSION P(3)
  COMMON /AB/ POLYGN(60,11),POLGN,SHAD(60)
  COMMON /AC/ EDGE1(100),EDGE2(100),EDGEN
  COMMON /AAA/ XE(120),YE(120),ZE(120),POINTN
  INTEGER POLYGN,POLGN,EDGE1,EDGE2,EDGEN,POINTN
  IF(J.EQ.11)GO TO 211
  NUM=POLYGN(IP,11)
  NTWO=J-1
  IF(J.EQ.1)NTWO=NUM
  L=POLYGN(IP,J)
  L2=POLYGN(IP,NTWO)
  M=EDGE1(L)
  M2=EDGE2(L2)
  XS(M)=P(1)
  XS(M2)=P(1)
  YS(M)=P(2)
  YS(M2)=P(2)
  ZS(M)=P(3)
  ZS(M2)=P(3)
  RETURN
211 POLYGN(IP,J)=P(1)
  RETURN
END

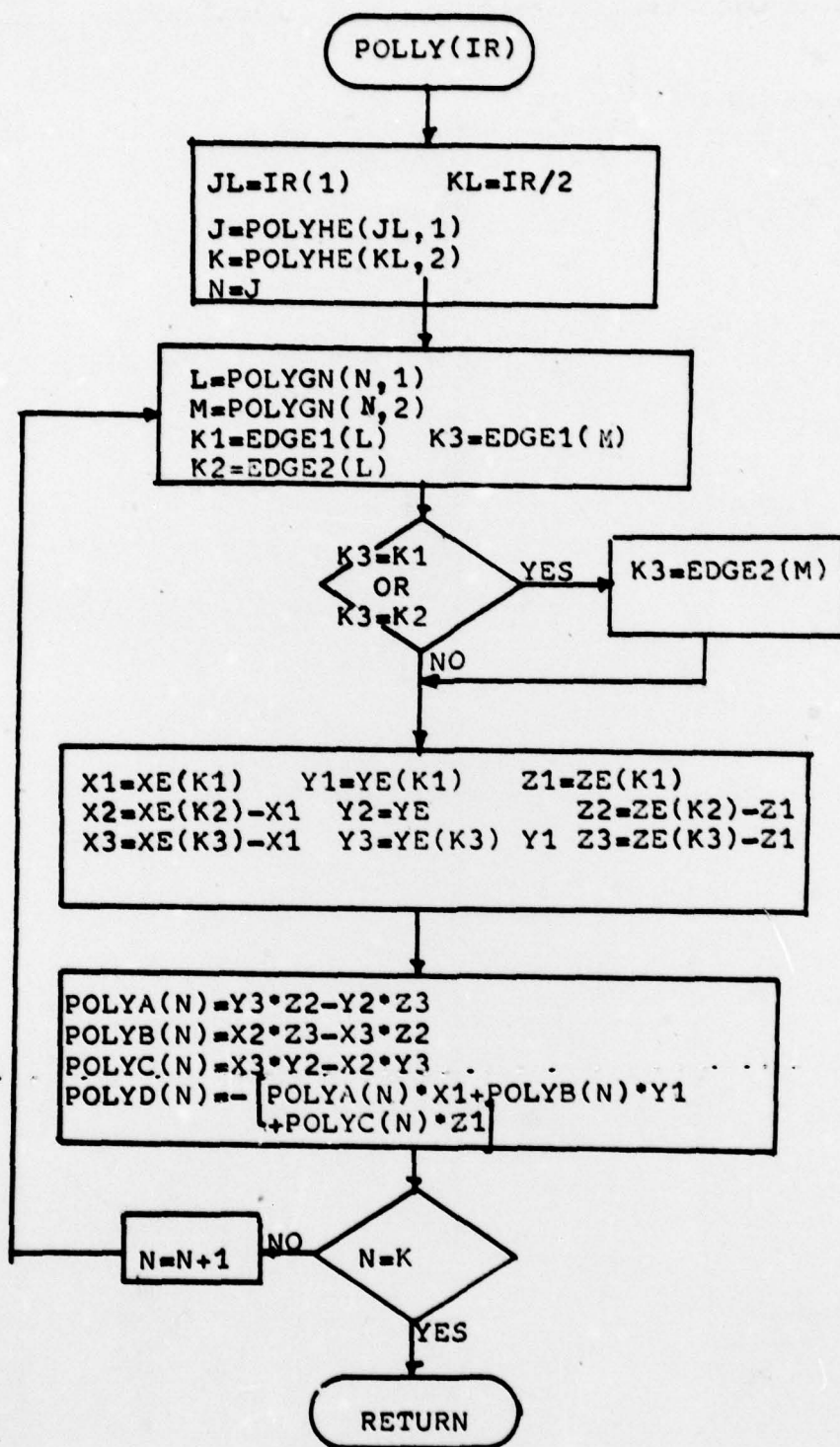
```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      POLLY: COMPUTES THE POLYGONAL PLANE COEFFICIENTS FOR A SET
C      OF POLYHEDRA AS DETERMINED BY THE CALLING PARAMETER IR.
C      IR(1) = THE INDEX OF THE FIRST POLYHEDRON
C      IR(2) = THE INDEX OF THE LAST POLYHEDRON
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE POLLY(IR)
      DIMENSION IR(2)
      COMMON /CA/ POLYA(60),POLYB(60),POLYC(60),POLYD(60)
      COMMON /AA/ POLYHE(10,2),POLYHN
      COMMON /AB/ POLYGN(60,11),POLGN
      COMMON /AC/ EDGE1(100),EDGE2(100),EDGEN
      COMMON /AA4/ XF(120),YF(120),ZE(120),POINTN
      INTEGER POLYHE,POLYHN,POLYGN,POLGN,EDGE1,EDGE2,EDGEN,POINTN
      JL=IR(1)
      KL=IR(2)
      J=POLYHE(JL,1)
      K=POLYHE(KL,2)
      DO 1910 N=J,K
        L=POLYGN(N,1)
        M=POLYGN(N,2)
        K1=EDGE1(L)
        K2=EDGE2(L)
        K3=EDGE1(M)
        IF(K3.EQ.K1.OR.K3.EQ.K2) K3=EDGE2(M)
        X1=XF(K1)
        Y1=YF(K1)
        Z1=ZE(K1)
        X2=XF(K2)-X1
        Y2=YF(K2)-Y1
        Z2=ZE(K2)-Z1
        X3=XF(K3)-X1
        Y3=YF(K3)-Y1
        Z3=ZE(K3)-Z1
        POLYA(N)=Y3*Z2-Y2*Z3
        POLYB(N)=X2*Z3-X3*Z2
        POLYC(N)=X3*Y2-X2*Y3
        POLYD(N)=- (POLYA(N)*X1+POLYB(N)*Y1+POLYC(N)*Z1)
1910 CONTINUE
      RETURN
      END

```



```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      SUMMIT: IS USED BY THE SUBROUTINE INTERR TO SUM THE X, Y,
C      AND Z VALUES FOR ALL OF THE VERTICES FOR THE CURRENT
C      POLYHEDRON.  ADDITIONALLY, A RUNNING COUNT OF THE
C      NUMBER OF VERTICES SUMMED IS MAINTAINED.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

      SUBROUTINE SUMMIT(K)
      COMMON /AAA/ XF(120),YE(120),ZE(120),POINTN
      COMMON /CH/ XSUM,YSUM,ZSUM,ICNT
      COMMON /FF/ PCHANG(200)
      INTEGER POINTN,PCHANG
      PCHANG(K)=1
      XSUM=XSUM+XE(K)
      YSUM=YSUM+YE(K)
      ZSUM=ZSUM+ZE(K)
      ICNT=ICNT+1
      RETURN
      END

```

```

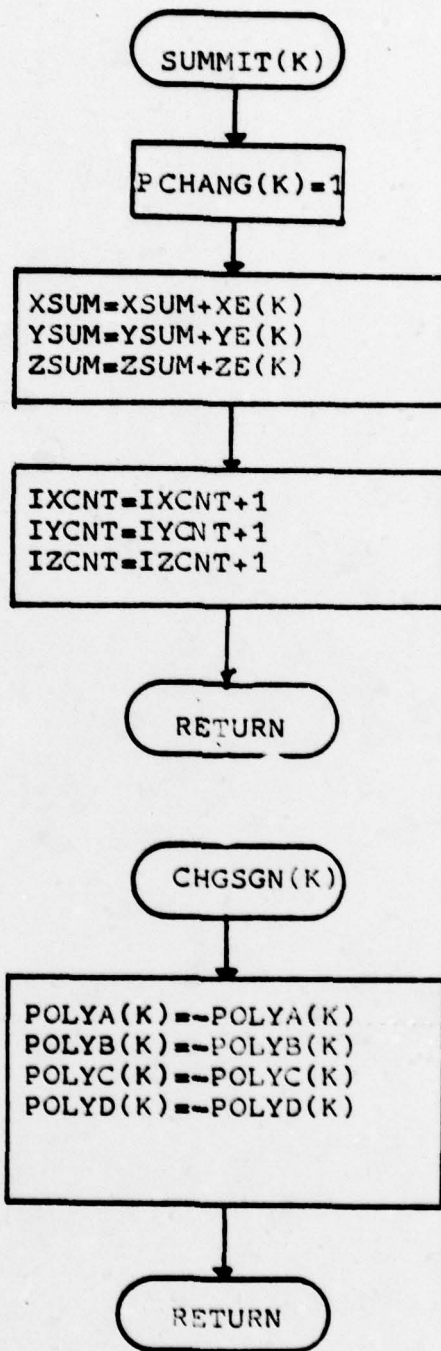
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      CSHAD: CHANGES THE SHADE OR COLOR OF ANY SET OF POLYHEDRONS.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

      SUBROUTINE CSHAD(IR,NEW)
      COMMON /AA/ POLYHE(10,2),POLYHN
      COMMON /AH/ POLYGN(60,11),POLGN,SHAD(60)
      DIMENSION IR(2)
      INTEGER POLYHE,POLYHN,POLYGN,POLGN,SHAD
      JL=IR(1)
      KL=IR(2)
      J=POLYHE(JL,1)
      K=POLYHE(KL,2)
      DO 720 I=J,K
        SHAD(I)=NEW
720  CONTINUE
      RETURN
      END

```



FCCTNOTES

<sup>1</sup> Phong, Bui Tuong, "Illumination for Computer Generated Pictures," Communications of the ACM, v. 16, no. 6, p.311, June 1975.

<sup>2</sup> Schumacker, Robert A., Sproull, Robert F., and Sutherland, Ivan E., "A Characterization of Ten Hidden-Surface Algorithms," Computing Surveys, v. 6, no. 1, p.9, March 1974.

<sup>3</sup> Newman, William A. and Sprcull, Robert F., Principles of Interactive Computer Graphics, McGraw-Hill, p. 258-262, 1973.

\* Ibid., p. 247.

\* Ibid., p. 256-259.

\* Ibid., p.249.

\* Ibid., p.316-317.

\* Ibid., p. 545.

\* Schumacker, p. 13-14.

# LIST OF REFERENCES

1. Newman, William M. and Sproul, Robert F., Principles of Interactive Computer Graphics, McGraw-Hill, 1973.
2. Appel, A. and Will, P. M., "Determining the Three-Dimensional Convex Hull of a Polyhedron", IBM Journal of Research and Development, p. 596-601, November 1976.
3. Blin, James F. and Newell, Martin E., "Texture and Reflection in Computer Generated Images", Communications of the ACM, v. 19, no. 10, p. 542-547, October 1976.
4. Braid, I. C., "The Synthesis of Solids Bounded by Many Faces", Communications of the ACM, v. 18, no. 4, p. 209-216, April 1975.
5. Clark, James H., "Hierarchical Geometric Models for Visible Surface Algorithms", Communications of the ACM, v. 19, no. 10, p. 547-554, October 1976.
6. Crow, Franklin C., "The Aliasing Problem in Computer-Generated Shaded Images", Communications of the ACM, v. 20, no. 11, p. 799-805, November 1977.
7. Graham, N. Y., "Perspective Drawing of Surfaces With Hidden Line Elimination", The Bell System Technical Journal, v. 51, no. 4, p. 843-861, April 1972.
8. Levin, Joshua, "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadratic Surfaces", Communications of the ACM, v. 19, no. 10, p. 555-563, October 1976.
9. Natsushita, Yutaka, "Hidden Line Elimination for a Rotating Object", Communications of the ACM, v. 15, no. 4, p. 245-252, April 1972.
10. McGrath, F. J., "A Method for Eliminating Hidden Lines With Polyhedra", Simulation, p. 37-41, January 1971.
11. Phong, Bui Tuong, "Illumination for Computer Generated Pictures", Communications of the ACM, v. 18, no. 6, p. 311-317, June 1975.
12. Schumacker, Robert A., Sproull, Robert A., and Sutherland, Ivan E., "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, v. 6, no. 1, p. 1-58, March 1974.
13. Teschler, Leland, "New Realism for Computer Graphics", Machine Design, p. 93-99, February 23, 1978.
14. Williamson, Hugh, "Hidden-Line Plotting Program", Communications of the ACM, v. 15, no. 2, p. 100-103, February 1972.

# INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Professor D. E. Kirk, Code 62Ki Chairman, Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4. Professor M. L. Cotton, Code 62Cc Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3
5. Professor R. Panholzer, Code 62 Pz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
6. Lt. Homer J. Rood, USN Rt. 8 Box 631 Pensacola, Florida 32506	1